THE UNIVERSITY OF NEW MEXICO

ECE 538
Advanced Computer Architecture

Instructor: Lei Yang

Department of Electrical and Computer Engineering

September 14, 2021

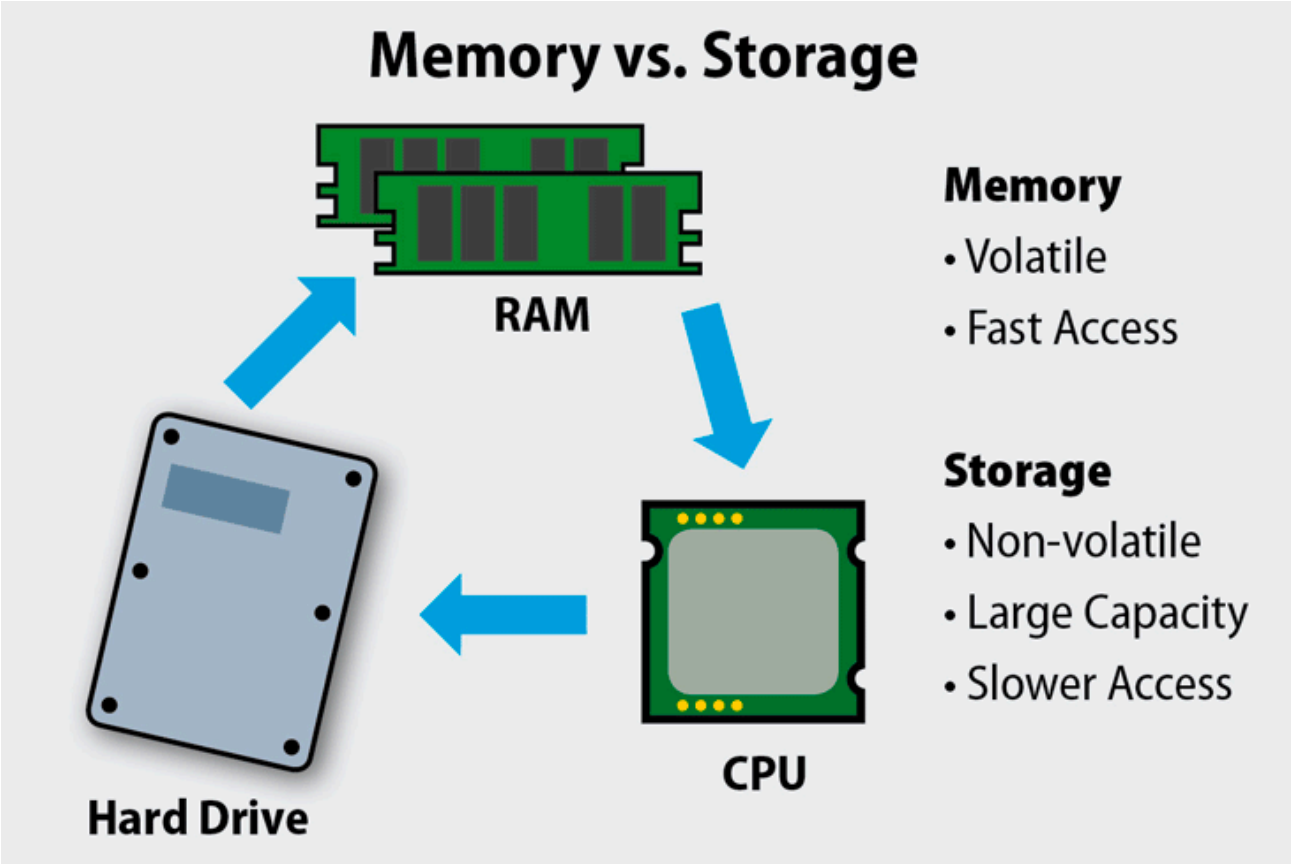
2021-**-**: End Class ** here

- Overview of the computer architecture
- Computer abstraction and techniques
- Performance metrics

Memory
in Computer Architecture

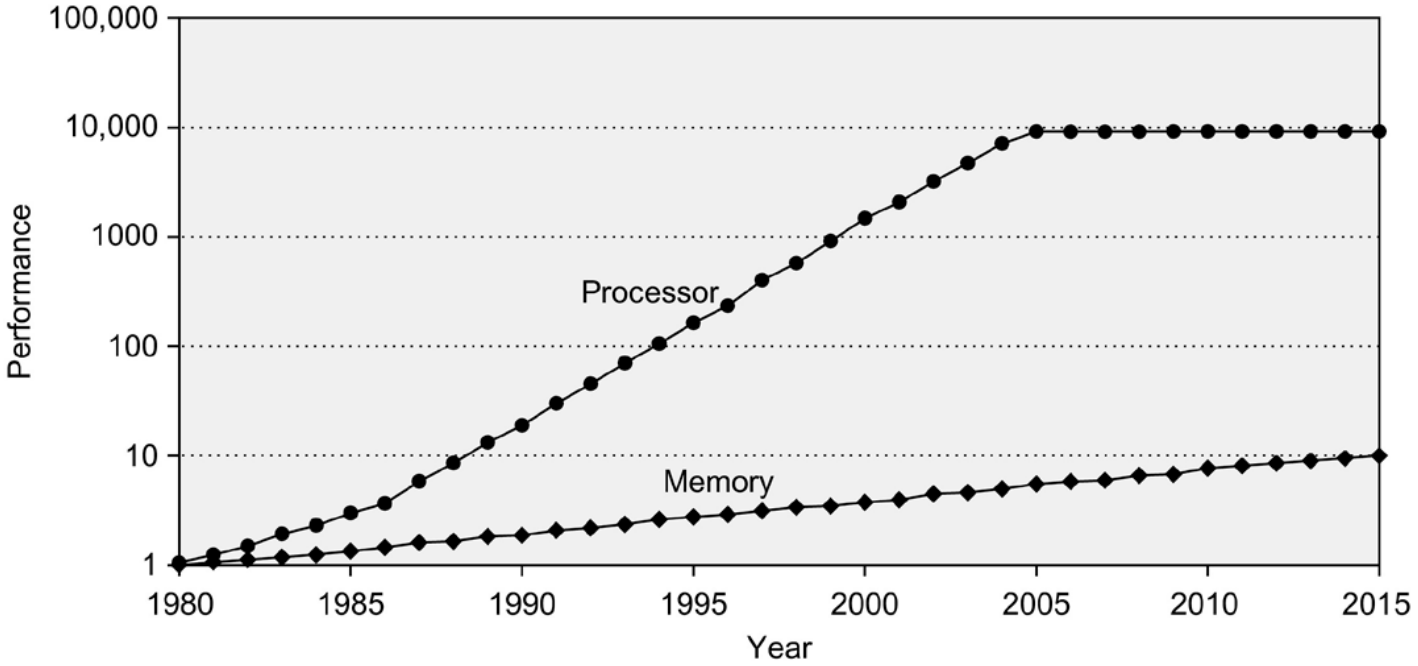
Referred to Chapter 2

MEMORY VS. STORAGE



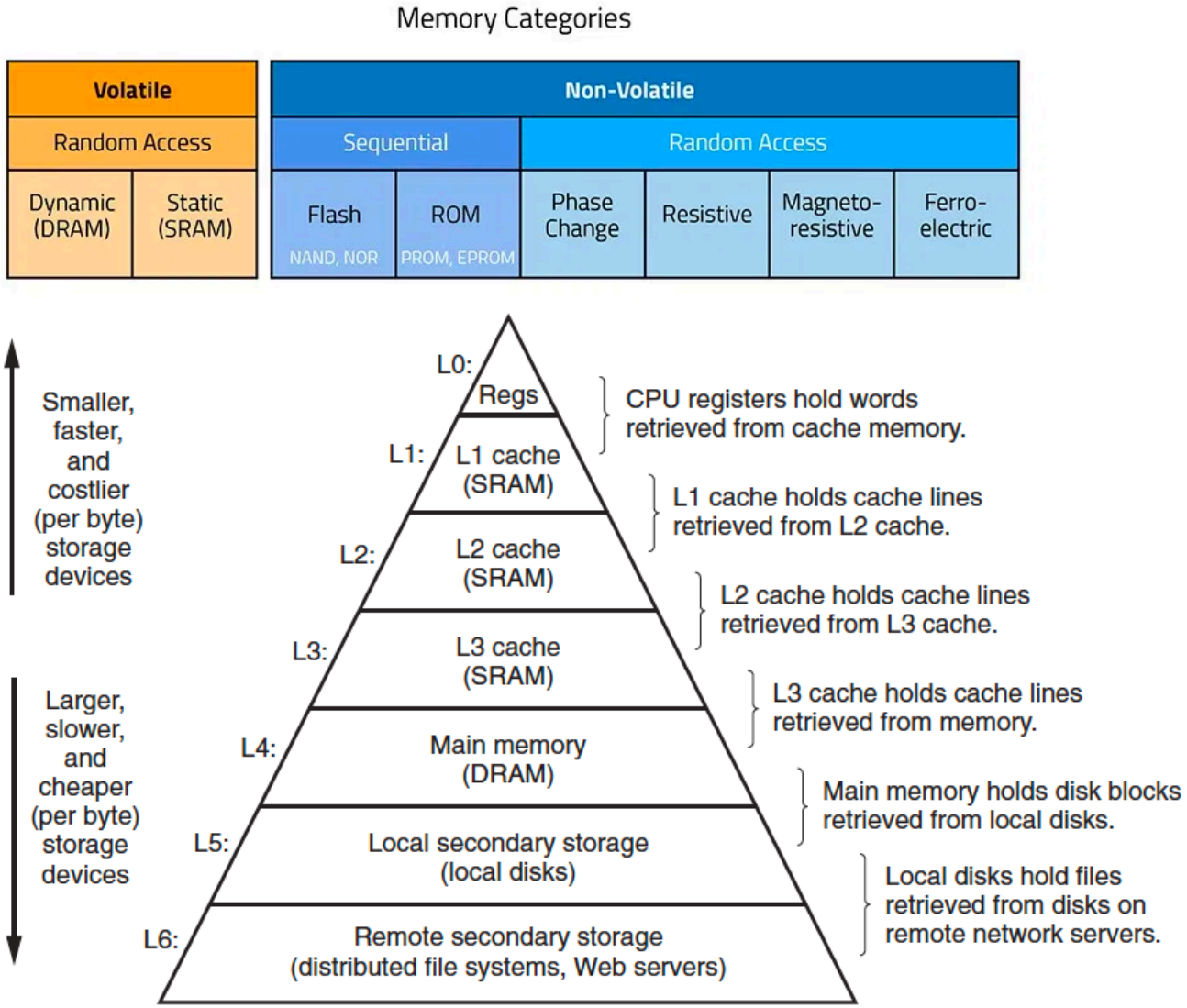
Click figure to watch the video

Processor / Memory Performance Gap

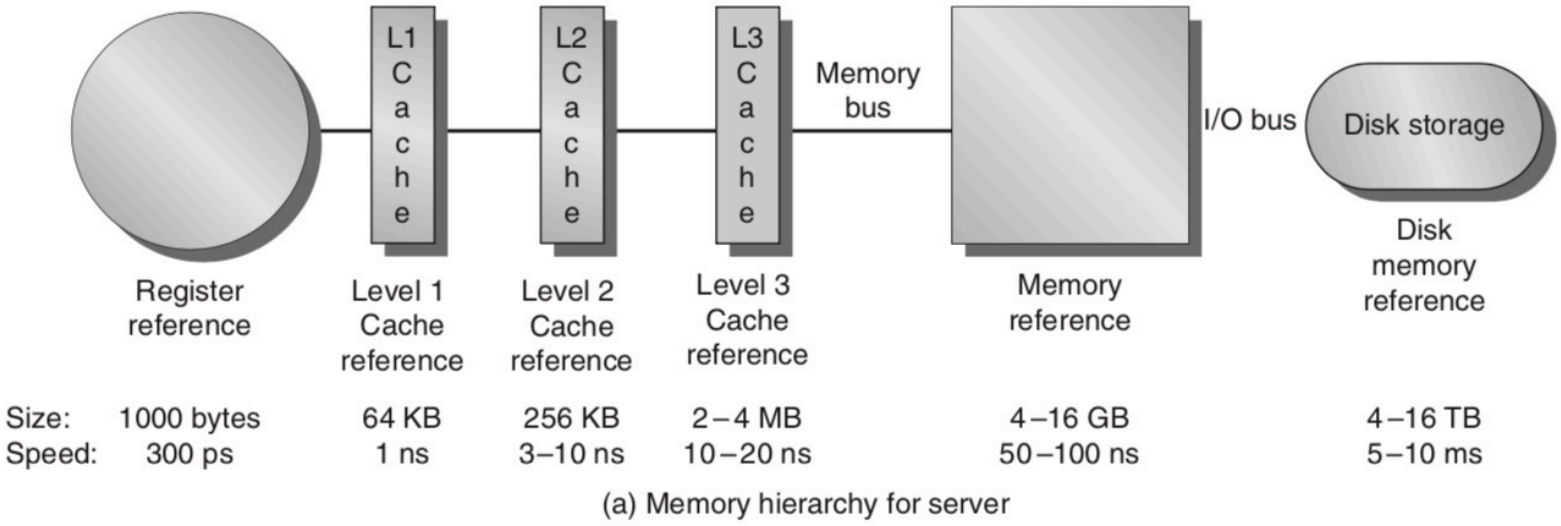


- Programmers want unlimited amounts of memory with low latency
- Processor performance has significantly outpaced memory performance
  - Size of main memory has increased dramatically, Larger memories are slower
  - DRAM is denser but slower compared to SRAM
  - Creates a gap in performance that must be filled

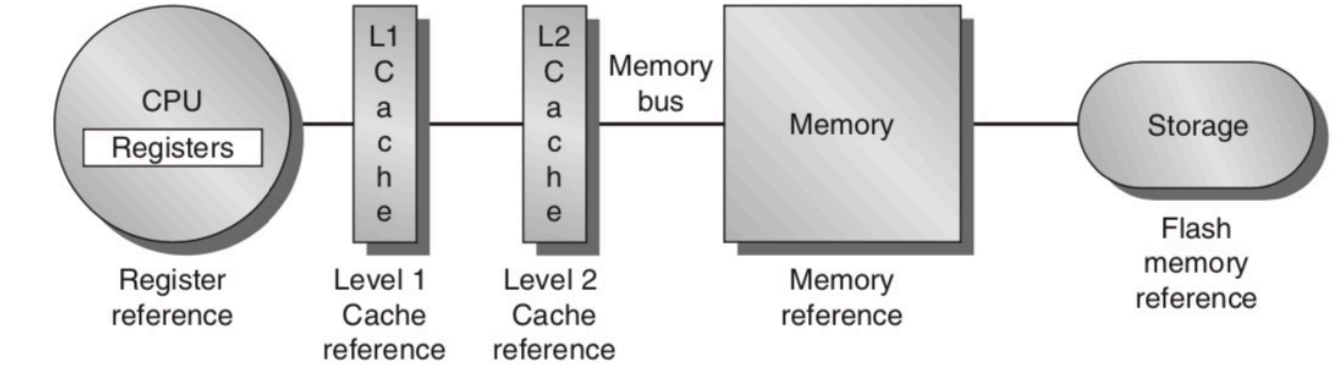
Memory Hierarchy Design



Memory Hierarchy Design

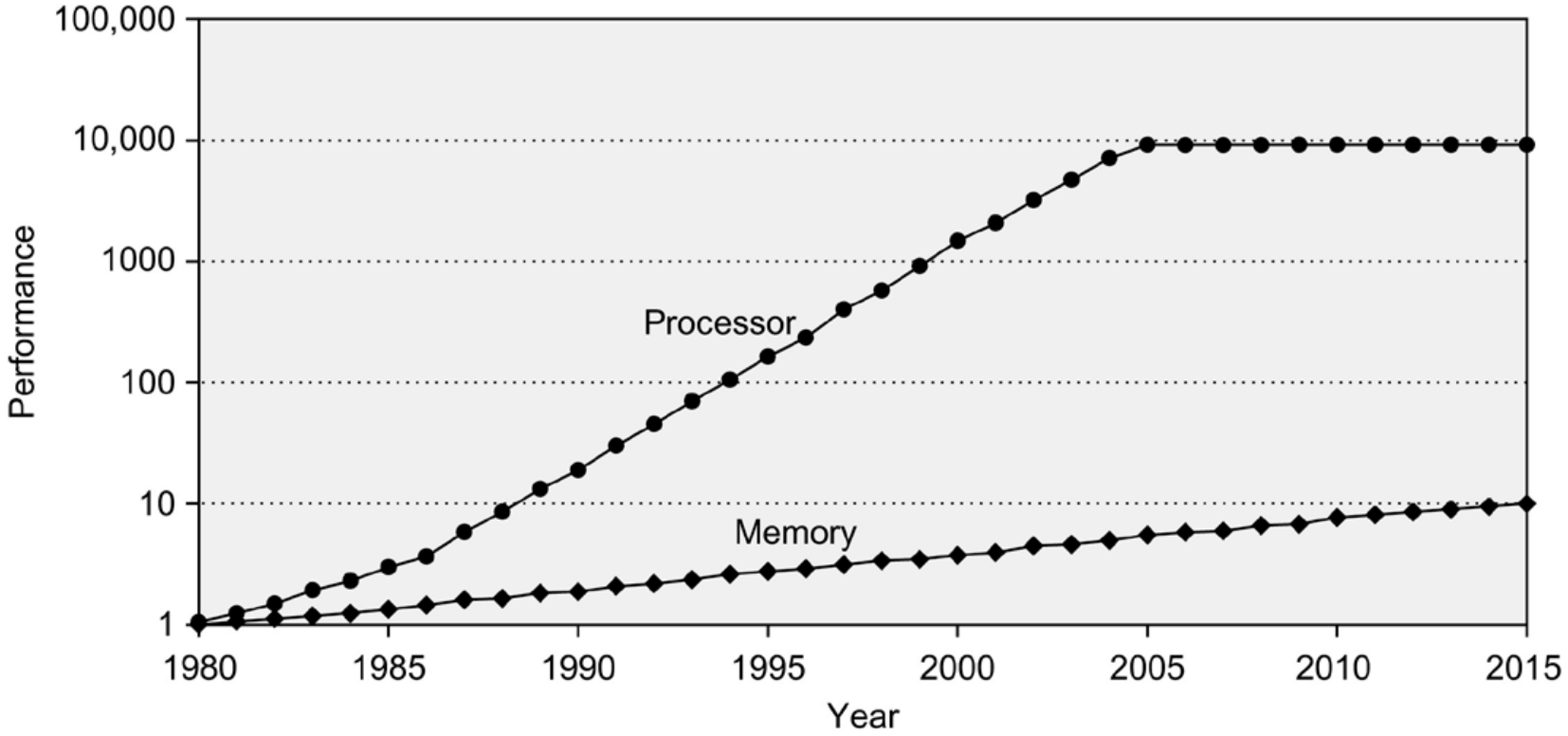


(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Processor / Memory Performance Gap



The gap in performance, measured as the difference in the time between processor memory requests (for a single processor or core) and the latency of a DRAM access, is plotted over time.

Memory Hierarchy Design

- Memory hierarchy design becomes more crucial in recent multi-core era
- Aggregate peak bandwidth requirements grow with the number of cores
  - Intel Core i7 can generate two data memory references per core per clock
  - Consider four cores and a 4.2GHz clock generates 33.6 billion 64-bit data references per second
  - The i7 also requires about 12.8 billion 128-bit instruction references per second
  - All this adds up to 441.1 GiB/s of memory bandwidth!
- DRAM alone can only support a bandwidth of 34.1GiB/s which is 8% of required bandwidth

Memory Hierarchy Design

- Average Memory Access Time (AMAT) used to be primary design metric

$$\textit{Average memory access time} = \textit{Hit time} + \textit{Miss rate} \times \textit{Miss penalty}$$

- Now: **Power** is a major concern
  - High-end microprocessors have > 10MiB of on-chip cache!
  - Caches burn power even while idle, i.e. leakage power
  - In mobile devices, 25% to 50% of power budget goes into cache

Memory Hierarchy Design

- Compiler manages registers
- Hardware manages caches

Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<4 KiB	32 KiB to 8 MiB	<1 TB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk or FLASH
Access time (ns)	0.1–0.2	0.5–10	30–150	5,000,000
Bandwidth (MiB/sec)	1,000,000–10,000,000	20,000–50,000	10,000–30,000	100–1000
Managed by	Compiler	Hardware	Operating system	Operating system
Backed by	Cache	Main memory	Disk or FLASH	Other disks and DVD

Basics of Memory Hierarchy

- A key design decision is where blocks (or lines) can be placed in a cache
- **Set Associative**: the set is chosen by the address of the data:

$$(\textit{Block address}) \text{ MOD } (\textit{Number of sets in cache})$$

- Caching data that is only read is easy; **Caching writes** is more difficult
- **Cache Miss Rate**: a measure of benefits of different cache organizations
  - Compulsory
  - Capacity
  - Conflict
- **Average memory access time:** better measure of cache organization

$$\textit{Average memory access time} = \textit{Hit time} + \textit{Miss rate} \times \textit{Miss penalty}$$

Basics of Memory Hierarchy

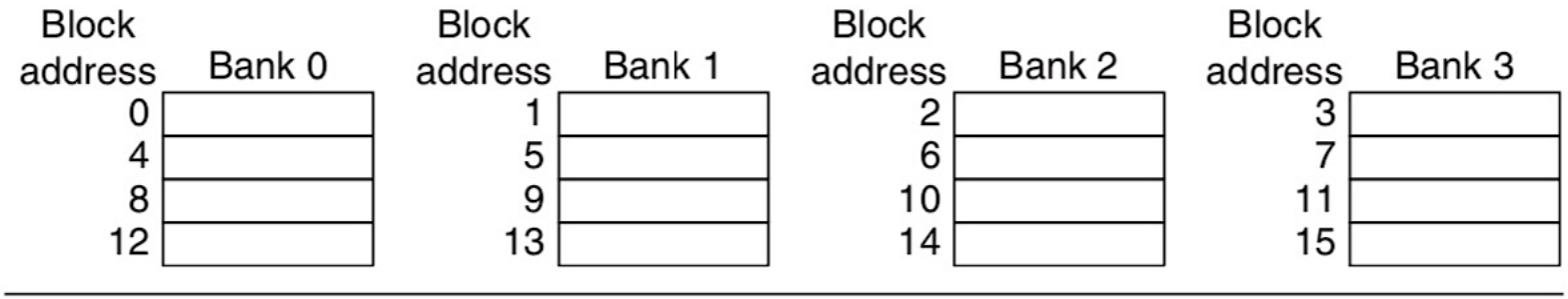
- **Six basic cache optimizations:**
  - Larger block size to reduce miss rate
  - Bigger caches to reduce miss rate
  - Higher associativity to reduce miss rate
  - Multilevel caches to reduce miss penalty
  - Giving priority to read misses over write misses to reduce miss penalty
  - Avoiding address translation during indexing of the cache to reduce hit time

Ten Advanced Optimizations of Cache Performance

- Five categories of the ten advanced optimizations:
  - Reducing the hit time
  - Increasing cache bandwidth
  - Reducing the miss penalty
  - Reducing the miss rate
  - Reducing the miss penalty or miss rate via parallelism

Ten Advanced Optimizations of Cache Performance

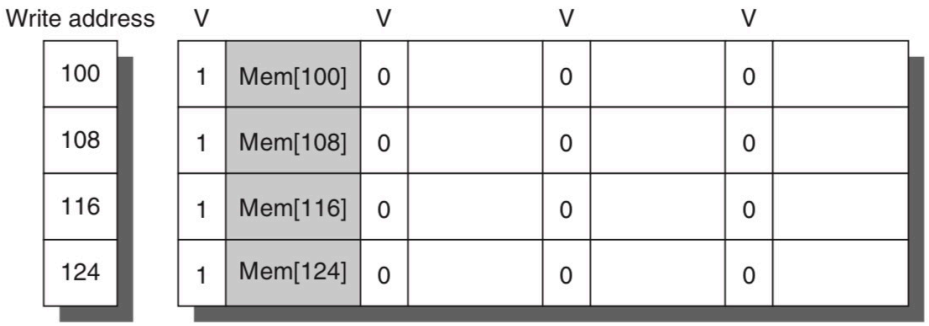
- Small and Simple First-Level Caches to Reduce Hit Time and Power
- Way Prediction to Reduce Hit Time
- Pipelined Cache Access to Increase Cache Bandwidth
- Nonblocking Caches to Increase Cache Bandwidth
  - Difficulty: a cache miss does not necessarily stall the processor
- Multibanked Caches to Increase Cache Bandwidth

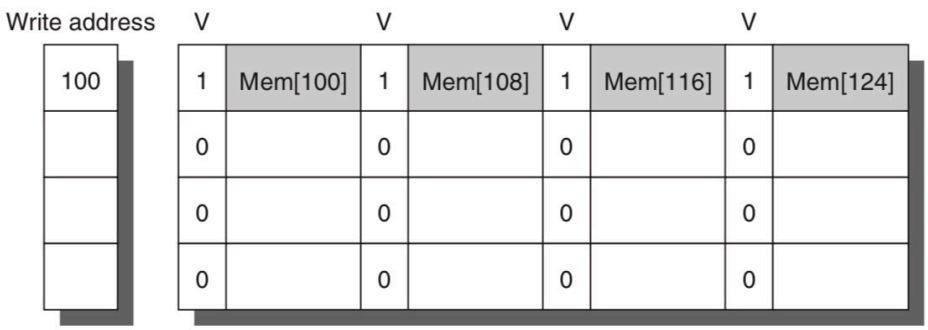


Four-way interleaved cache banks using block addressing

Ten Advanced Optimizations of Cache Performance

- Critical Word First and Early Restart to Reduce Miss Penalty
  - **Critical word first**—Request missed word first from memory and send it to processor as soon as it arrives; let processor continue execution while filling the rest of the words in the block.
  - **Early restart**—Fetch words in normal order, but as soon as requested word of block arrives send it to processor and let processor continue execution.
- Merging Write Buffer to Reduce Miss Penalty





To illustrate write merging, the write buffer on top does not use it while the write buffer on the bottom does

Ten Advanced Optimizations of Cache Performance

- Compiler Optimizations to Reduce Miss Rate
  - Loop Interchange
  - Blocking
- Hardware Prefetching of Instructions and Data to Reduce Miss Penalty or Miss Rate
- Compiler-Controlled Prefetching to Reduce Miss Penalty or Miss Rate
  - Register prefetch will load the value into a register.
  - Cache prefetch loads data only into the cache and not the register.

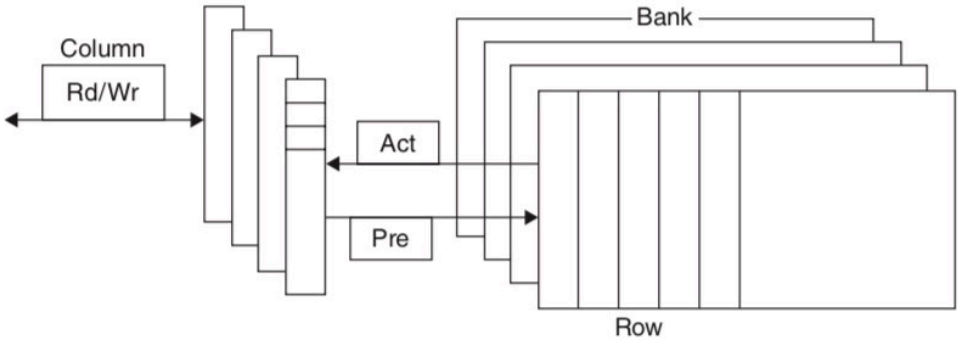
CACHE OPTIMIZATION SUMMARY

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Memory Technology and Optimizations

- SRAM Technology:
  - SRAM don't need to refresh, so the access time is very close to the cycle time
  - SRAM use six transistors/bit to prevent information from being disturbed when read
  - SRAM needs only minimal power to retain the charge in standby mode.

- DRAM Technology
  - Modern DRAMs are organized in banks, typically four for DDR3.
  - Each bank consists of a series of rows.
  - Each command, as well as block transfers, are synchronized with a clock.



Internal organization of a DRAM

Memory Technology and Optimizations

- Improving Memory Performance Inside a DRAM Chip
  - DRAMs added timing signals that allow repeated accesses to the row buffer without another row access time.
  - DRAMs had an asynchronous interface to the memory controller, so every transfer involved overhead to synchronize with the controller

    It is called Synchronous DRAM (SDRAM)
  - DRAMS were made wider for w wide stream of bits from memory
  - Double Data Rate (DDR)
- Reducing Power Consumption in SDRAMs
- Flash Memory
  - A type of EEPROM (Electronically Erasable Programmable Read-Only Memory)
  - Flash memory is normally read-only but can be erased.
- Enhancing Dependability in Memory Systems

- Flash memory
  - Flash uses a very different architecture and has different properties than standard DRAM. The most important differences are
    1. Flash memory must be erased before overwritten
    2. Flash memory is static and draws significantly less power when not reading or writing
    3. Flash memory has a limited number of write cycles for any block, typically at least 100,000.
    4. High-density Flash is cheaper than SDRAM but more expensive than disks:
    5. Flash is much slower than SDRAM but much faster than disk.

Crosscutting Issues: Fundamental of Memory Hierarchies

- Protection and Instruction Set Architecture
  - E.g. to support virtual memory in IBM 370, architects had to change successful IBM 360 instruction set architecture that had been announced just 6 years before. Similar adjustments are being made today to accommodate virtual machines.
- Coherency of Cached Data
  - multiple processors and I/O devices raise the opportunity for copies to be inconsistent and to read the wrong copy.

Crosscutting Issues: Memory Hierarchy Design

- Four Memory Hierarchy Questions?
  - 🔎 Where can a block be placed in the cache?
  - 🔎 How is a block located in the cache?
  - 🔎 Which block should be replaced on a miss?
  - 🔎 What happens on a write?

Crosscutting Issues: Memory Hierarchy Design

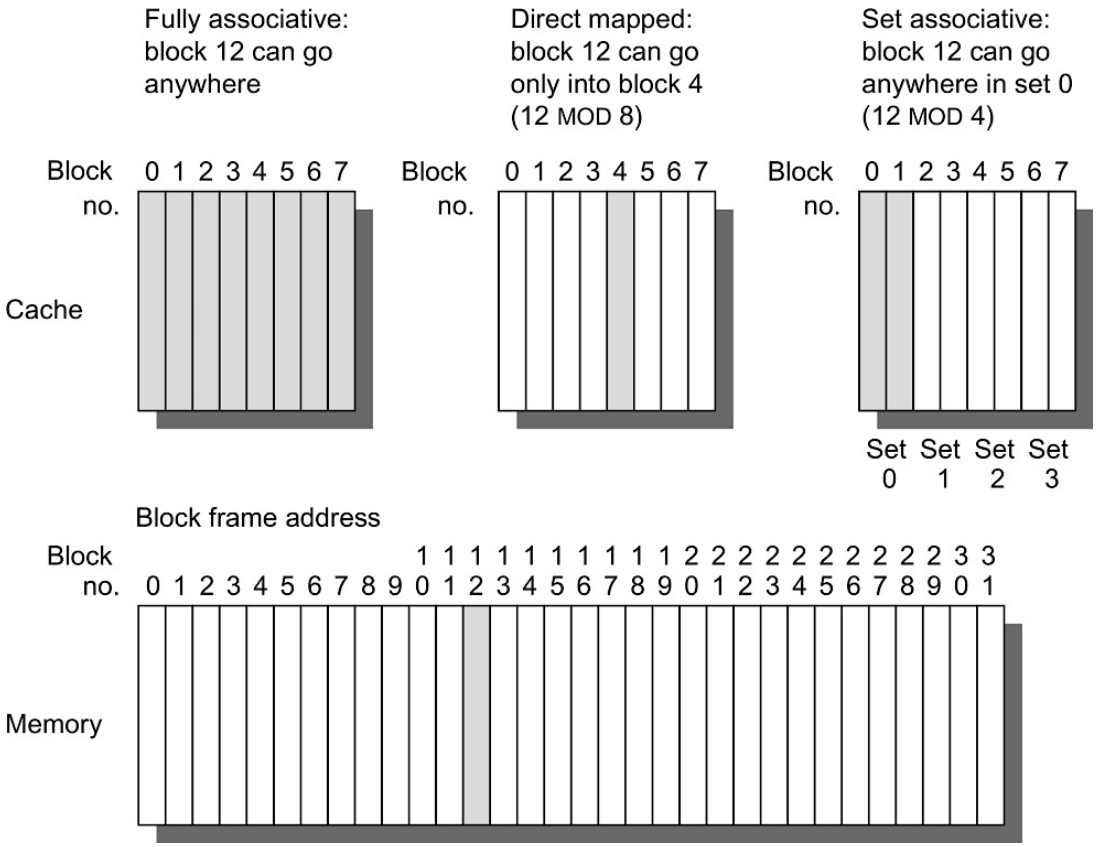
- When a block can be placed? – **Cache placement**
  - Direct-mapped
    - A block in memory can only go in one location in the cache
    - Rigid placement strategy makes hardware fast and simple
    - Causes ***conflict misses***

    …
  - Fully-associative
    - A block in memory can go anywhere in the cache
    - More complicated and slower

    …

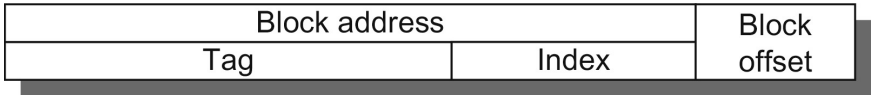
Memory Hierarchy Design

- The restrictions on where a block is placed create three categories of cache organization



- The cache is broken up into sets
- Within each set are ways
- A block in memory can go anywhere within a given set
- Example: A 2-way set-associative cache has two locations within a set that a block can go into
- Not as slow and complicated as fully associative
- Does not suffer from as many conflict misses as direct-mapped
- 2- and 4- way are typical

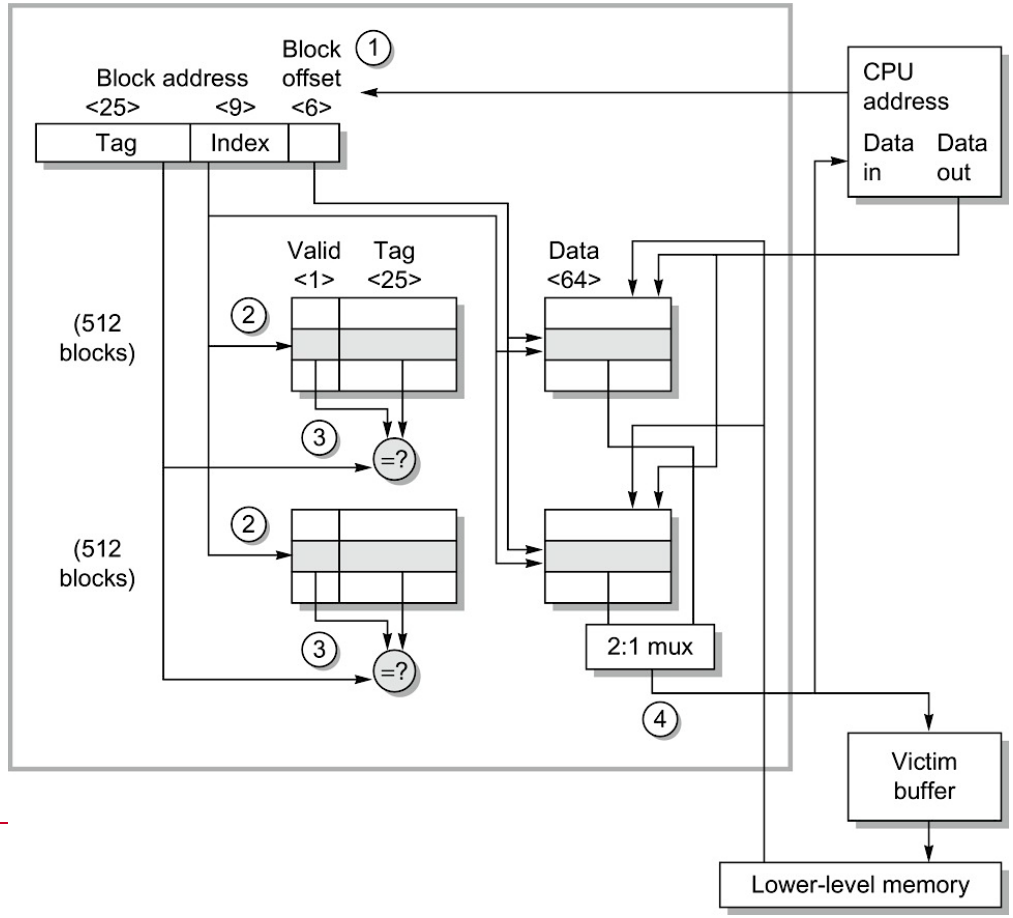
Memory Hierarchy Design



- How is a memory block located?
  - Have a way to uniquely identify a block in the cache
  - A tag identifies the block within the set
  - An index identifies the set

- Example: *Opteron Data cache*

- Caches are constructed from SRAMs, multiplexers, and comparators
- Need a valid bit
- Let's draw a simple direct-mapped cache on the board
- We can combine direct-mapped caches to make set-associative caches assuming the associativity is low (2 or 4)



Memory Block Replacement and Associativity

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KiB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KiB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KiB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Data cache misses per 1000 instructions comparing least recently used, random, and first in first out replacement for several sizes and associativities.

- Observations from the table
  - Replacement strategy matters much less with larger caches
  - LRU does best, then FIFO, then random but all pretty close!
  - Associativity also matter less with larger caches
  - More significant decrease from 2-way to 4-way
  - If direct-mapped was shown, there would be an even more significant decrease from direct-mapped to 2-way

Write Operations in Memory

- Around 27.8% of data cache traffic are writes
  - Data shows 10% stores and 26% loads for RISC-V
  - This equates to 7.4% of overall memory traffic
- We must consider how data is written into the L1 cache as well as the memory or L2, etc.
  - *write-through*: Data is written to the L1 and memory (L2) on every write
  - *write-back*: On every write, data is written to the L1 only, and is only written to memory (L2) on eviction
- For write-back, the cache must keep track of when a block was modified so it knows to write it back to the next level upon eviction
- This is done with a dirty bit
- Write buffers can reduce unnecessary stalling on writes, especially for write-through
- On a write without a prior read:

  Write allocate: block is allocated in the cache immediately
  No write allocate : Block is simply forwarded to the next level

Memory Block Replacement and Associativity

- Which block should be replaced?
  - Inevitably blocks are going to conflict
  - **For direct-mapped caches**, the replacement strategy is simple
  - **For set- and fully-associative caches**, we have multiple ways within a set that a block can go and thus must have a replacement strategy:
    - *Random:* select a block at random to evict
    - *Least Recently Used (LRU):* evict the LRU block
    - *First In First Out (FIFO):* approximates LRU, because the LRU is difficult with higher associativity

Miss Rate and AMAT

- As block size increases, so does the miss penalty
- Thus, miss rate does not tell the whole story!
- Average Memory Access Time (AMAT) incorporates miss rate, miss penalty, and hit time

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} * \text{Miss penalty}$$

		Cache size			
Block size	**Miss penalty**	**4K**	**16K**	**64K**	**256K**
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

- Average Memory Access Time in clock cycles
- Shows 64 byte as the clear winner for 16kB and greater

The Types of Cache Miss

- The three C's
  - Compulsory – Initial misses due to a cold cache
  - Conflict – Misses due to lack of associativity, i.e. too rigid of a placement strategy
  - Capacity – Misses due to the small cache size

- Mitigation of misses
  - Compulsory misses can be decreased with larger block sizes
  - Conflict misses can be decreased due to associativity
  - Capacity misses can be decreased with larger caches

Cache size (KiB)	Degree associative	Total miss rate	Miss rate components (relative percent) (sum = 100% of total miss rate)					
			Compulsory		Capacity		Conflict	
4	1-way	0.098	0.0001	0.1%	0.070	72%	0.027	28%
4	2-way	0.076	0.0001	0.1%	0.070	93%	0.005	7%
4	4-way	0.071	0.0001	0.1%	0.070	99%	0.001	1%
4	8-way	0.071	0.0001	0.1%	0.070	100%	0.000	0%
8	1-way	0.068	0.0001	0.1%	0.044	65%	0.024	35%
8	2-way	0.049	0.0001	0.1%	0.044	90%	0.005	10%
8	4-way	0.044	0.0001	0.1%	0.044	99%	0.000	1%
8	8-way	0.044	0.0001	0.1%	0.044	100%	0.000	0%
16	1-way	0.049	0.0001	0.1%	0.040	82%	0.009	17%
16	2-way	0.041	0.0001	0.2%	0.040	98%	0.001	2%
16	4-way	0.041	0.0001	0.2%	0.040	99%	0.000	0%
16	8-way	0.041	0.0001	0.2%	0.040	100%	0.000	0%

Reducing Miss Penalty

Reducing Miss Penalty

- Miss rate is only part of the AMAT equation
- As the performance gap increased, miss penalties between the L1 cache and memory became prohibitive
- Multi-level caches solved that problem

Multi-level Caches

- Sandwich a larger L2 cache between the L1 and memory
- Allows L1 to be small to optimize hit time
- L2 is large enough to have a relatively low miss rate but not too large to keep access time down
- L2 access time is approximately L1 miss penalty

Miss Rate for L2

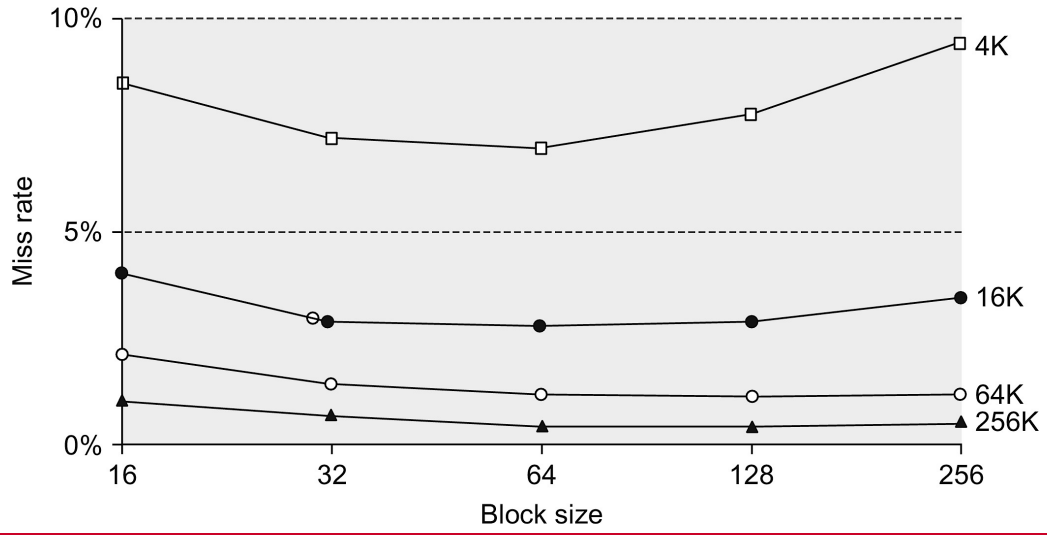
- Local miss rate
  - The number of misses divided by the number of access to that particular cache
  - Is often quite low for L2 cache because L1 handles most of the accesses and L2 is only accessed when L1 misses
- Global miss rate
  - The number of misses divided by the total number of memory accesses generated by the processor
  - For L1, global miss rate equals the local miss rate
  - For L2, global miss rate equals the local miss rate of L1 times the local miss rate of L2

TECHNIQUES THAT REDUCE MISS RATES

1. Larger Block Size to Reduce Miss Rate

- Larger blocks take advantages of
  - The bandwidth provided by the next level in the hierarchy
  - Spatial locality

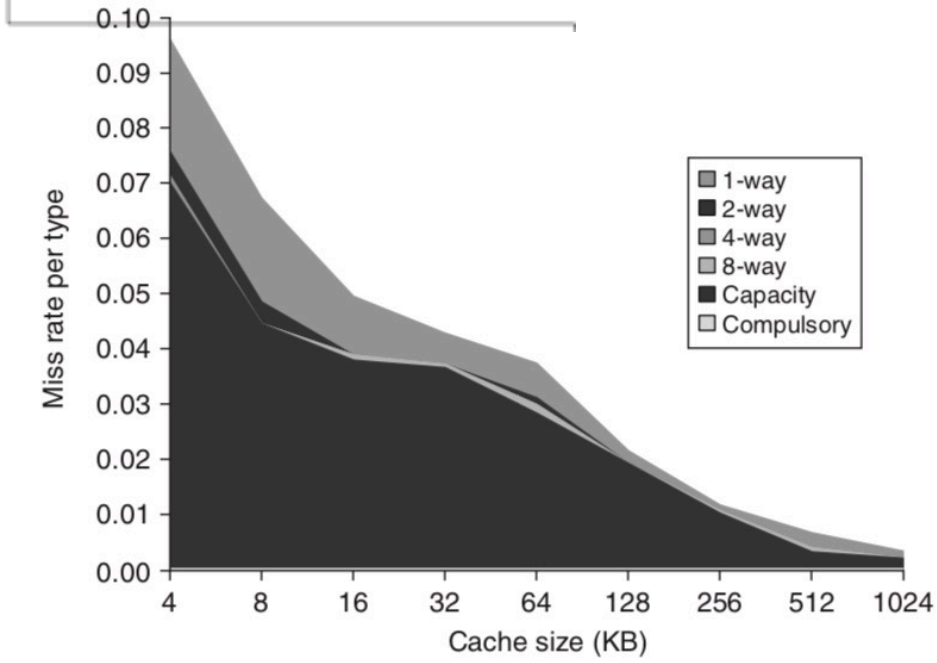
- Larger blocks reduce compulsory misses
- Block sized that are too large will start to suffer from conflict misses. *Why*?

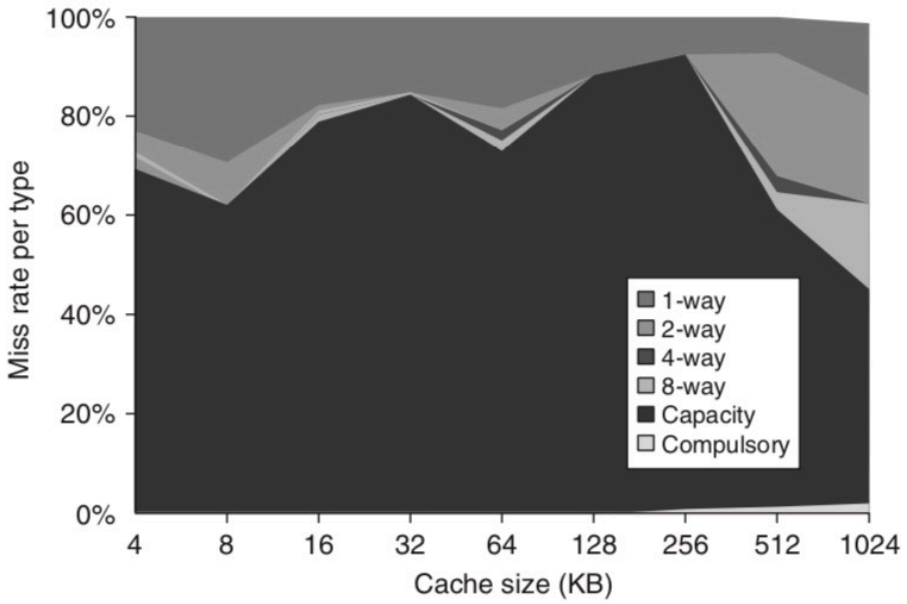


- Block size matters less for larger caches
- 64 bytes seems to be the happy place

TECHNIQUES THAT REDUCE MISS RATES

2. Larger Caches to Reduce Miss Rate



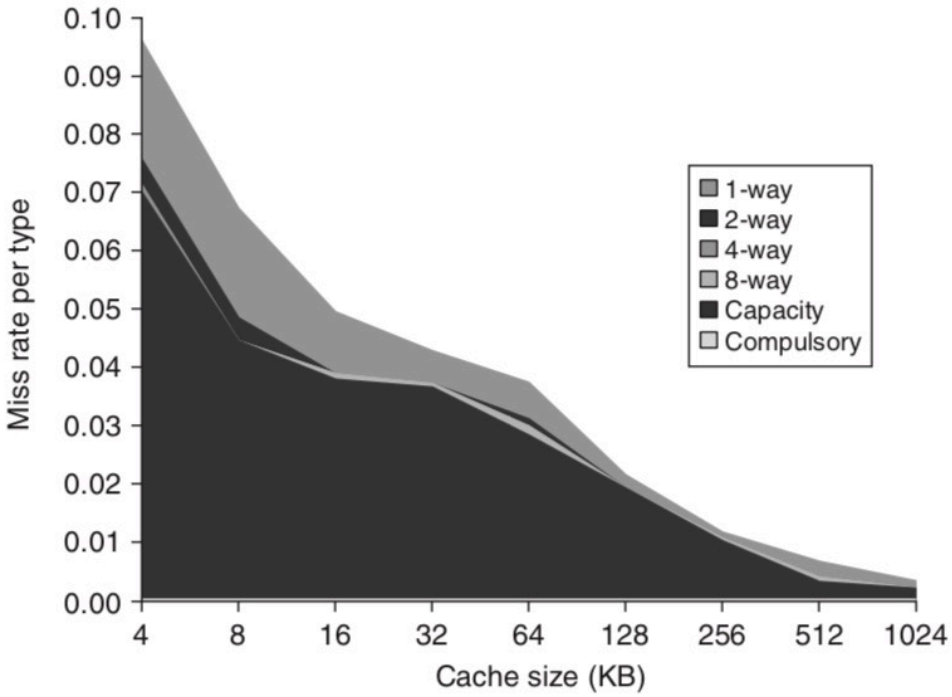


The obvious drawback is potentially longer hit time and higher cost and power. This technique has been especially popular in off-chip caches.

TECHNIQUES THAT REDUCE MISS RATES

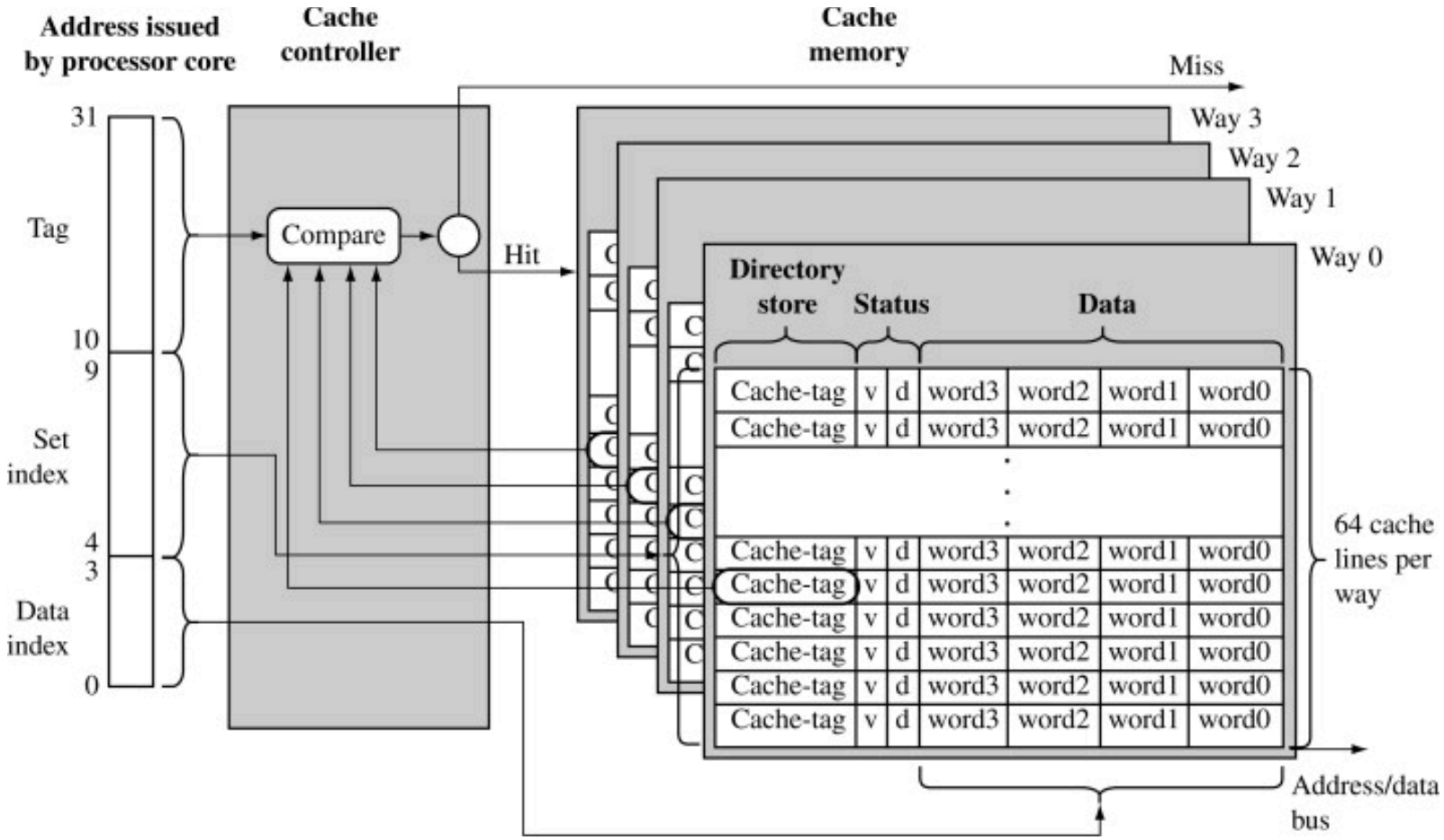
3. Higher Associativity to Reduce Miss Rate

- **Observation**: difference by comparing eight-way entries to capacity miss column in this Figure, since capacity misses are calculated using fully associative caches.
- **Observation**: called 2:1 cache rule of thumb, is that a direct-mapped cache of size N has about the same miss rate as a two-way set associative cache of size N/2. This held in three C's figures for cache sizes less than 128 KB



Associativity Caches

- Associativity Caches

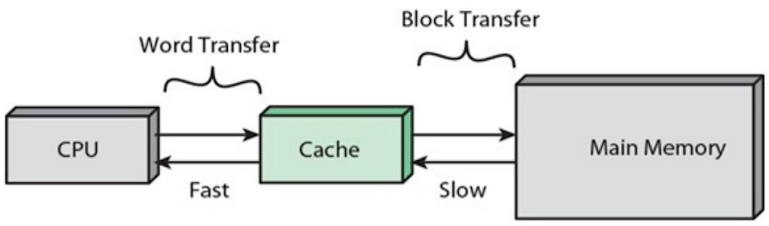


TECHNIQUES THAT REDUCE MISS RATES

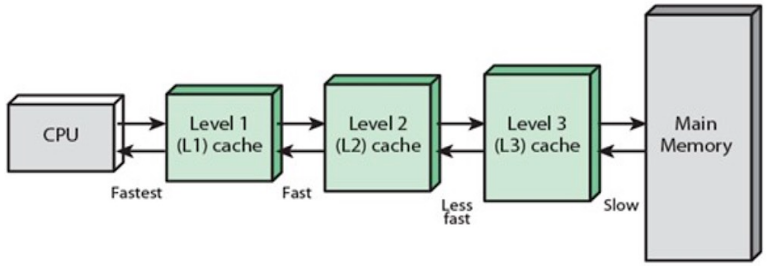
4. Multilevel Caches to Reduce Miss Rate

- Should I make cache faster to keep pace with speed of processors?
- Or make the cache larger to overcome the widening gap between the processor and main memory?

- To avoid the ambiguity, these terms are adopted here for a two-level
  - Local miss rate—This rate is simply the number of misses in a cache divided by the total number of memory accesses to this cache.
  - Global miss rate—The number of misses in a cache divided by the total number of memory accesses generated by the processor.



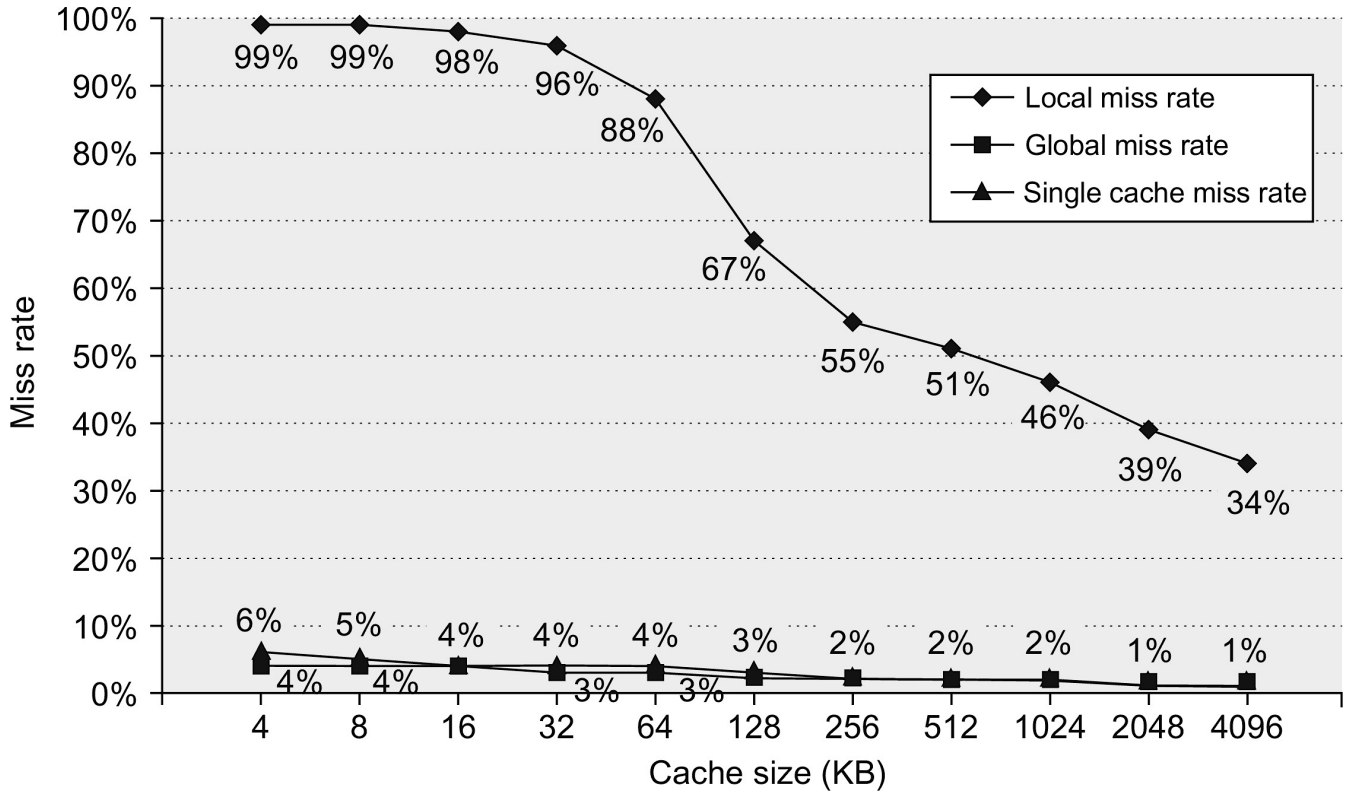
Single Cache



Three-level Cache Organization

Simulation Experiment

- Comparison of miss rate of multiple systems
  - Use Alpha architecture executing SPEC2000 benchmarks
  - Multilevel has split L1 each 64KiB, 2-way set associative Block sizes 64 bytes



Global Miss Rate:

- Is a better metric to use when discussing multilevel caches
- An indication of the fractions of accesses to cache system that go to main memory
- Track miss rate of a larger single-level cache system

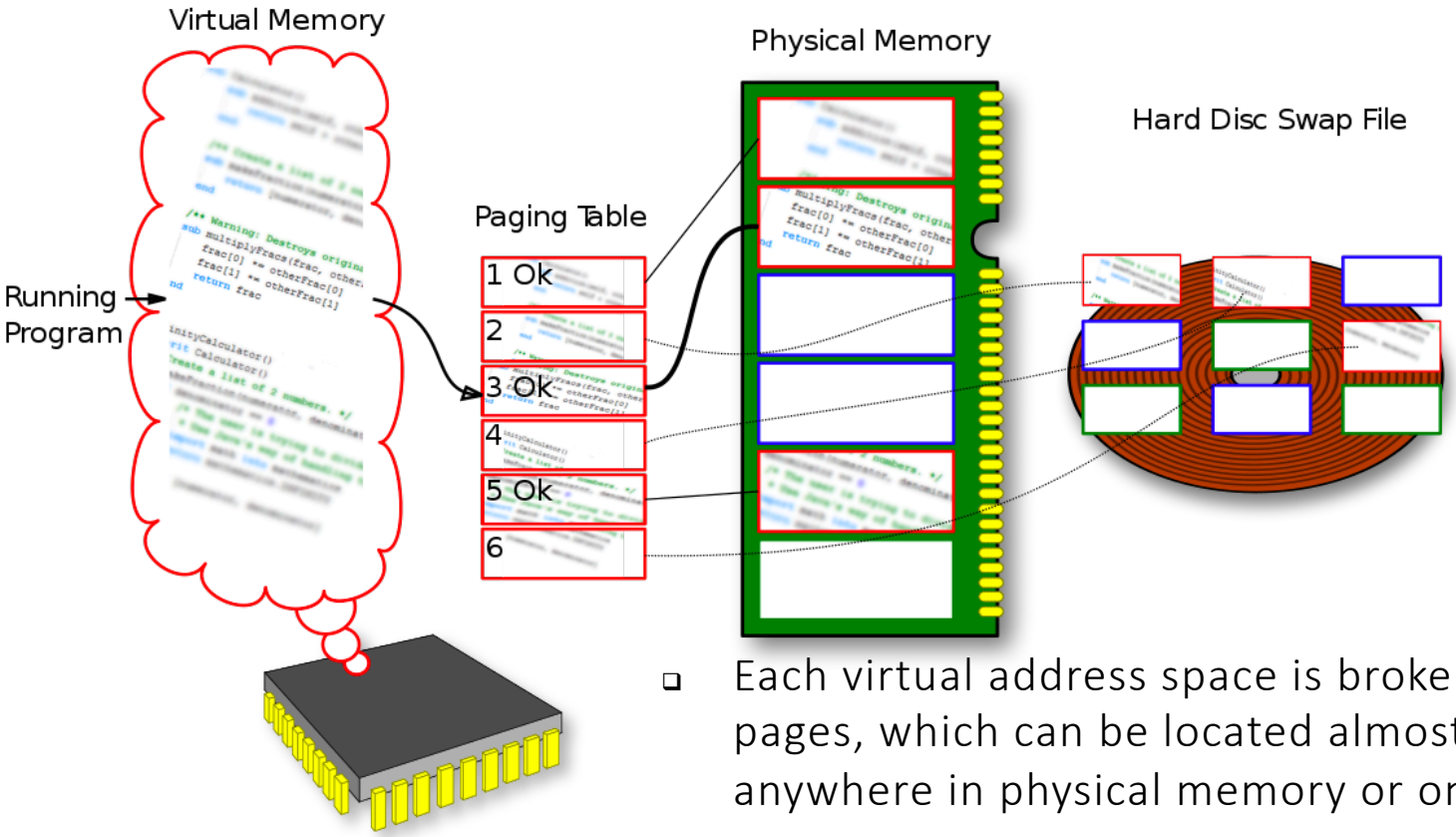
TECHNIQUES THAT REDUCE MISS RATES

5. Giving priority to read misses over writes to reduce Miss Penalty
6. Avoiding address translation during indexing of cache to reduce Hit Time

- Virtual Address is not built for all. *WHY?*
  - Protection: Page-level protection
  - Cache is to be flushed
  - Duplicate addresses, called synonyms or aliases
  - I/O typically uses physical addresses

VIRTUAL MEMORY:

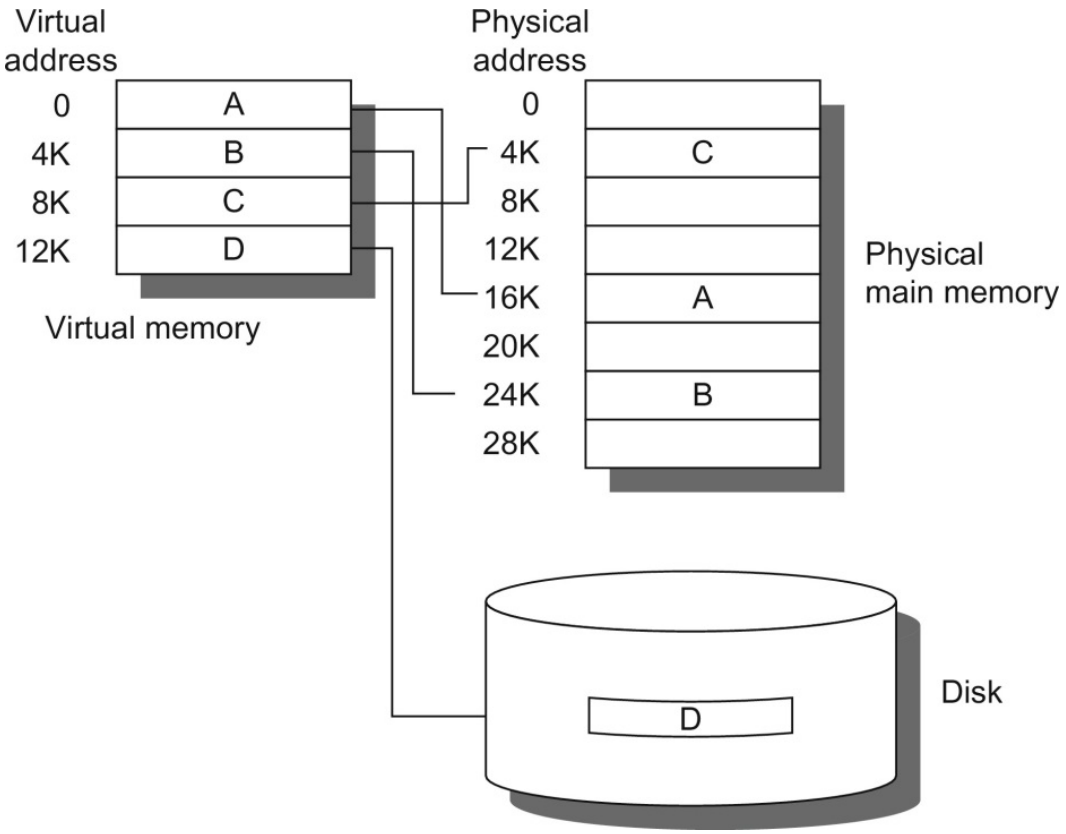
- Virtual memory provides an automatic mapping from virtual addresses of the program to physical addresses of the machine



- Each virtual address space is broken up into pages, which can be located almost anywhere in physical memory or on disk

- Storing pages on disk allows a program's virtual address space to be larger than the machines physical address space!

A Mapping of Address



The mapping of virtual memory to physical memory for a logical program

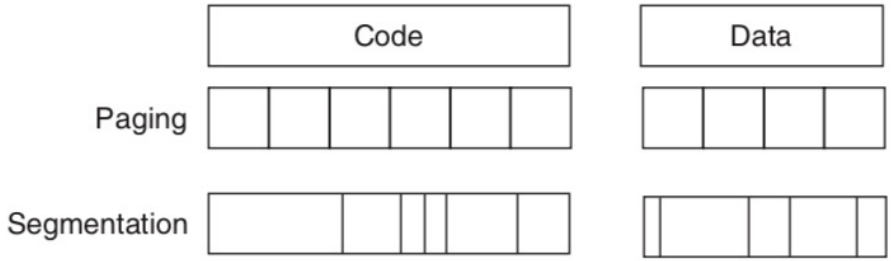
DIFFERENCES BETWEEN CACHES AND VIRTUAL MEMORY

Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–3 clock cycles	100–200 clock cycles
Miss penalty	8–200 clock cycles	1,000,000–10,000,000 clock cycles
(access time)	(6–160 clock cycles)	(800,000–8,000,000 clock cycles)
(transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.1–10%	0.00001–0.001%
Address mapping	25–45-bit physical address to 14–20-bit cache address	32–64-bit virtual address to 25–45-bit physical address

- Replacement on cache misses is primarily controlled by hardware, while virtual memory replacement is primarily controlled by operating system.
- The size of processor address determines the size of virtual memory, but the cache size is independent of the processor address size.

Two Categories of Virtual Memory

- **Pages**: fixed-size blocks
- **Segments**: variable-size blocks



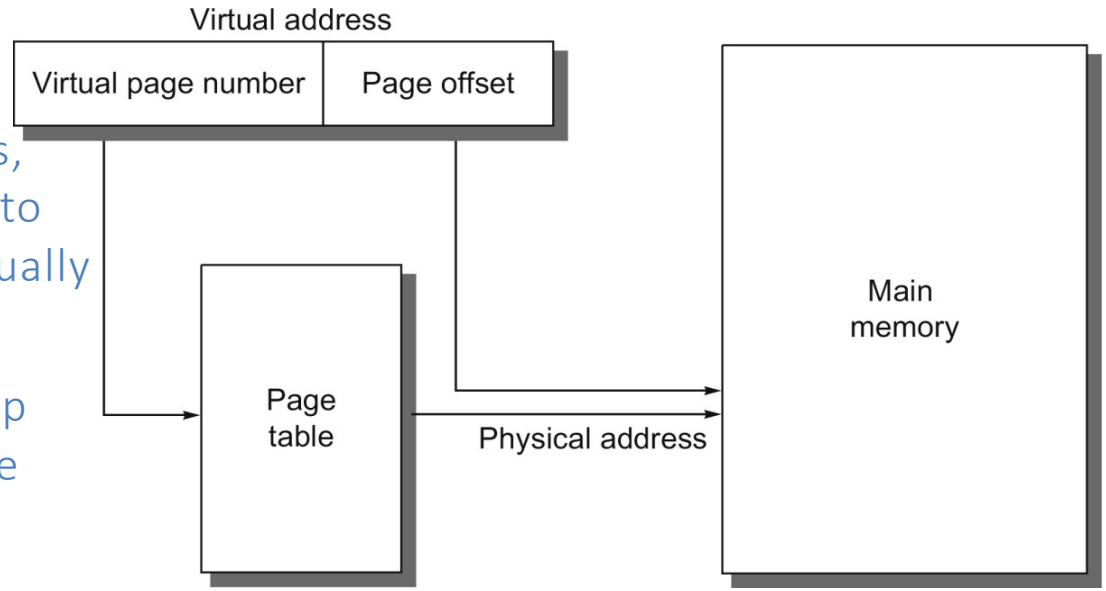
	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Difficult (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	Internal fragmentation (unused portion of page)	External fragmentation (unused pieces of main memory)
Efficient disk traffic	Yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes)

----- One hybrid approach: ***paged segments***

Multiple Virtual Address Spaces

- Address mapping (*address translation*) allow multiple virtual address memories
- Each process has its own virtual memory!
- This provides memory protection from one virtual address space to another, i.e. process A cannot access process B's memory unless it is specifically shared

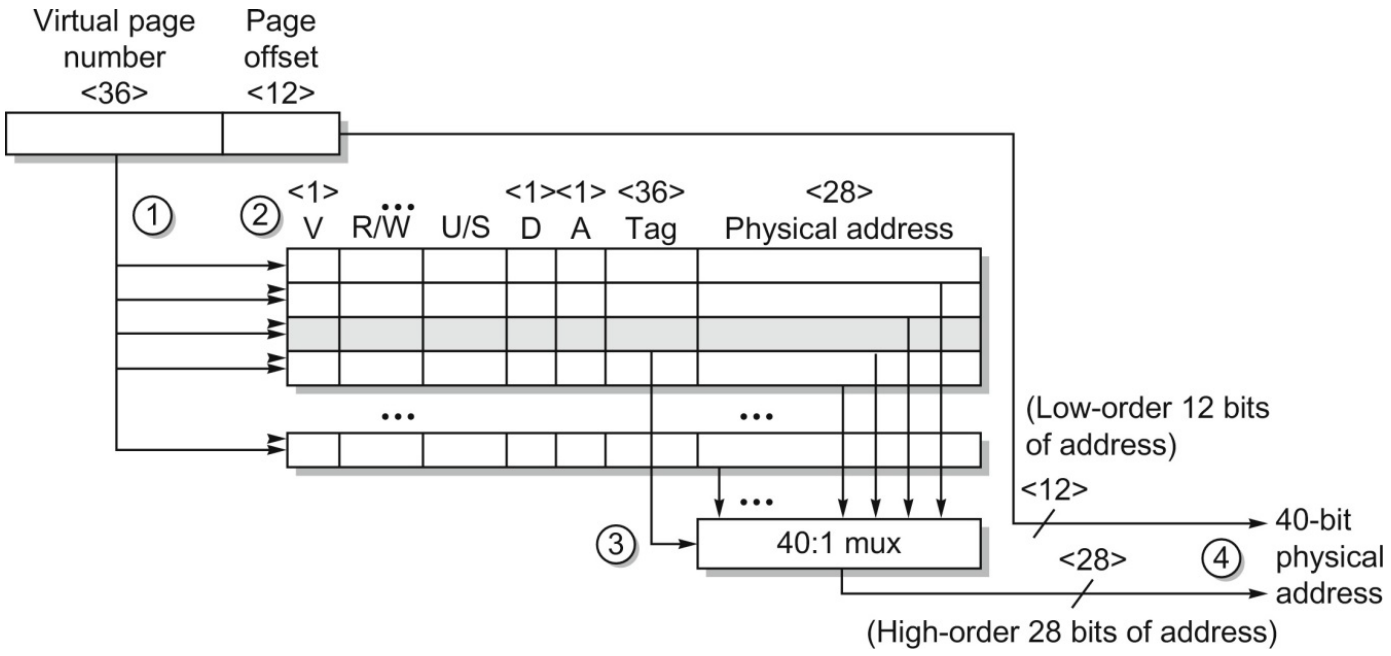
  - To efficiently translate addresses, virtual address space is divided into pages (typically 4kB in size but usually adjustable depending on OS)
  - Thus, virtual address is broken up into virtual page number and page offset
  - Virtual page number is then translated into a physical page number



The mapping of a virtual address to a physical address via a page table.

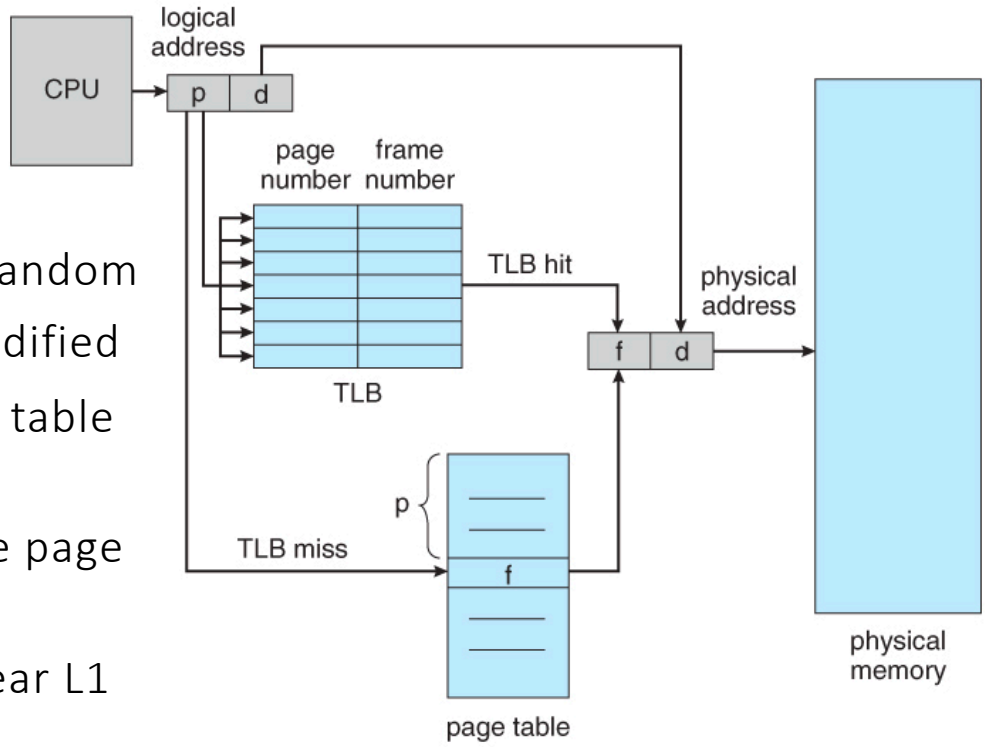
Multiple Virtual Address Spaces

- Making translation fast
  - Virtual memory would cause a significant slow down if the page table had to be accessed every time
  - Thus, we use a Translation Lookaside Buffer (TLB) to cache address translations
  - The TLB is maintained by the OS and uses the processor's exception logic to help with spilling and filling



More on Translation Lookaside Buffer (TLB)

- Usually has high associativity — fully associative is not uncommon
- Not very large — 40 -96 entries
- Contains protection bits such as read-write-execute
- Uses a Process ID (PID) in addition to a virtual page number to identify a physical page number



- Replacement strategy is commonly random
- Dirty bit indicates page has been modified
- TLB is filled by the OS from the page table during a TLB miss exception
- Must be capable of handling variable page sizes although 4KiB is typical
- Integrated into memory hierarchy near L1

Multiple Virtual Address Spaces

Physical vs. virtual addressing

- The L2 and beyond are physically addressed, i.e. after TLB
- But what about the L1?
  - Ideally the L1 would also be physically addressed to avoid aliasing
  - Some older pipelines would perform translation in a stage before cache access
  - Physically addressed L1 increases hit time
  - Virtually indexed, physically tagged is typically used

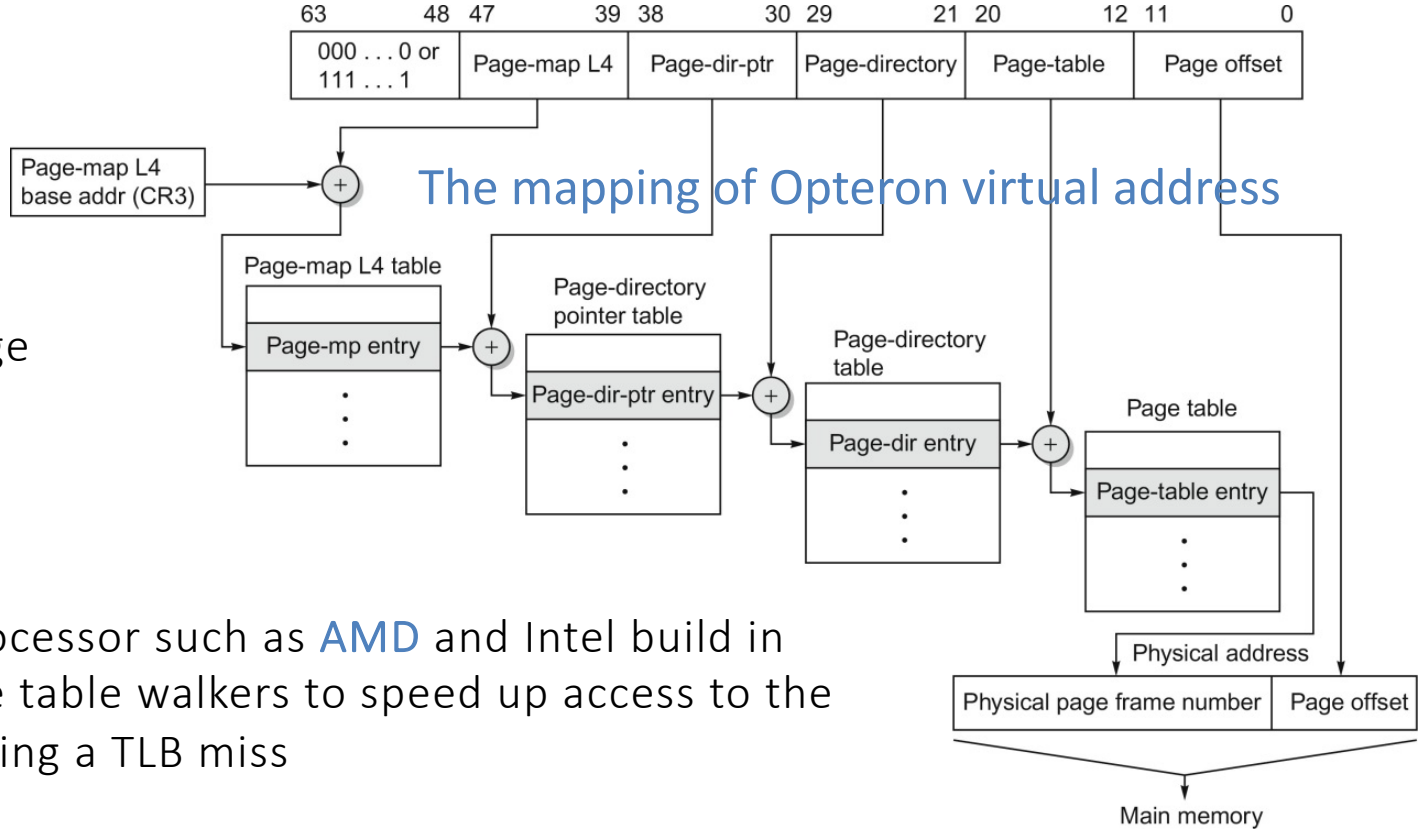
Virtually indexed, physically tagged

- Allows the lower part of the virtual address to index the cache
- The L1 and TLB can then be accessed in parallel!
- Aliasing is still possible but is either managed by OS to L2
- If the cache index is smaller than the page offset, no aliasing so this is where associativity helps! Why?

A Paged Virtual Memory

- Example: 64-Bit Opteron Memory Management

- Each process gets its own page table
- Page tables can get prohibitively large especially in a 48- to 64-bit address space typical of a modern 64-bit processor
- Most page tables are organized as multi-level page tables to manage size



- Higher end processor such as AMD and Intel build in hardware page table walkers to speed up access to the page table during a TLB miss

SEGMENTATION VS. PAGING

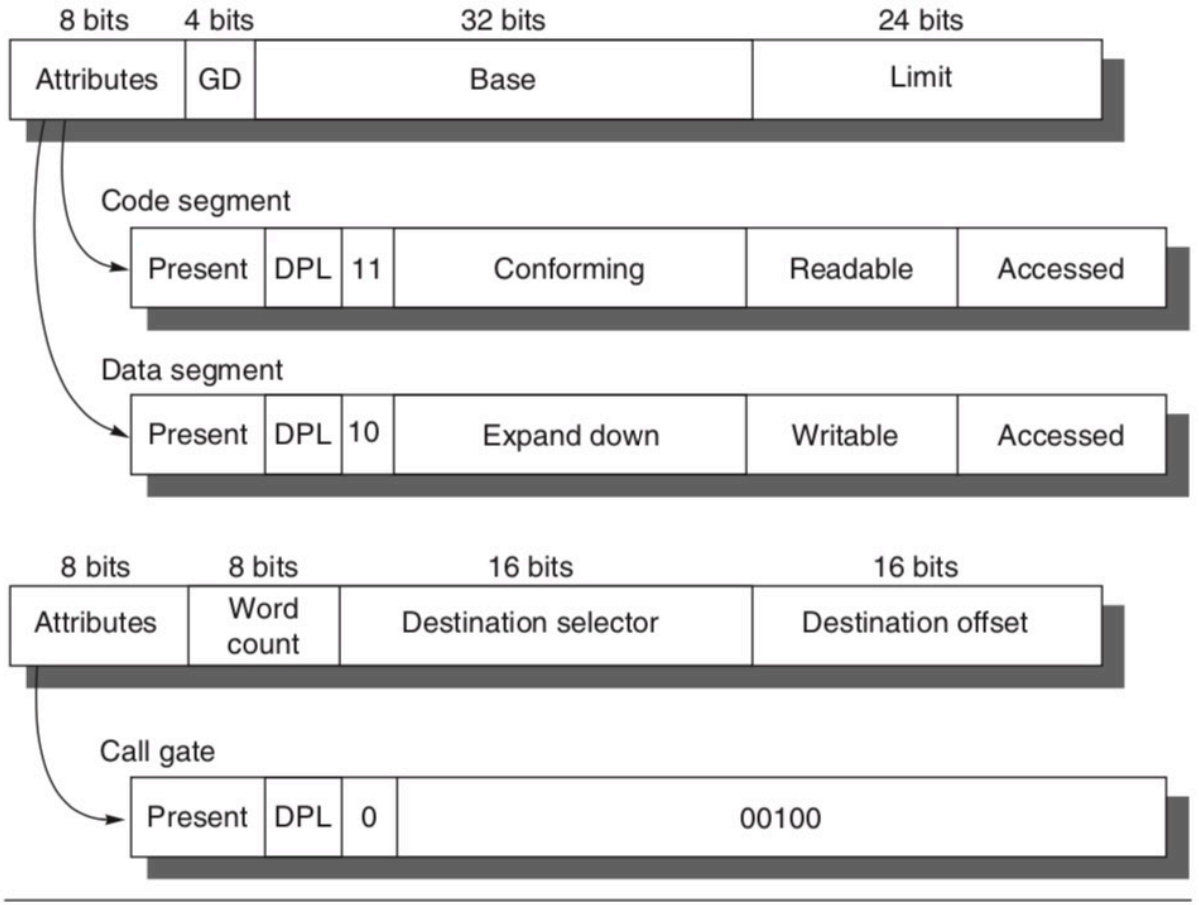
- So far we have looked at a technique, paging, for virtual memory
- An alternative method is segmentation
  - Memory space is divided into segments which can be variable in size rather than fixed like in paging
  - 2 words are used to describe an address location, segment and offset
  - x86 processors support both, while RISC tend to only support paging

	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Difficult (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	Internal fragmentation (unused portion of page)	External fragmentation (unused pieces of main memory)
Efficient disk traffic	Yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes)

A Segmented Virtual Memory

- Example: Protection in the Intel Pentium

- Adding Bounds Checking and Memory Mapping: get segmented addressing to check bounds as well as supply a bas
- Adding sharing and protection
- Adding Safe Calls from User to OS Gates and Inheriting Protection Level for Parameters



IA-32 segment descriptors are distinguished by bits in attributes field

VIRTUAL MACHINE (VM)

A virtual machine is taken to be an efficient, isolated duplicate of the real machine. We explain these notions through the idea of a virtual machine monitor (VMM). . . . a VMM has three essential characteristics.

-- Gerald Popek and Robert Goldberg (1974)

- Supports isolation and security in modern systems
- Less failures in security and reliability
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making overhead more acceptable
- Different ISAs and operating systems can be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Note: The software that supports VMs is called a *virtual machine monitor* (VMM) or *hypervisor*

Impact of VM on Virtual Address and I/O

- Each guest OS in every VM maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called “real memory”
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - Requires VMM to detect guest’s changes to its own page table
    - Occurs naturally if accessing the page table pointer is a privileged operation
- To virtualize the TLB in many RISC computers
- System virtualization due to the increasing number of I/O devices
- Sharing of a real device among multiple VMs

Cache Optimizations
in Computer Architecture

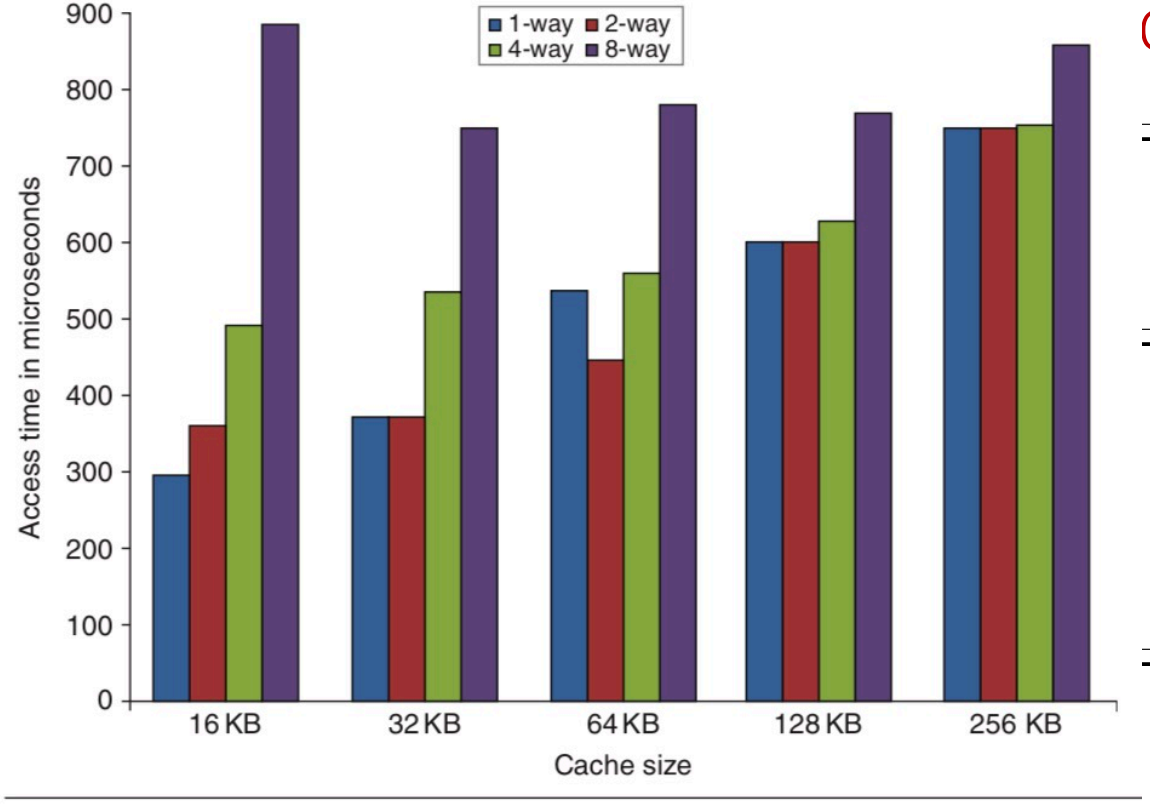
Advanced Cache Optimizations

- Reducing hit time – small and simple first-level and way-prediction
- Increasing cache bandwidth – pipelining caches, multi-banked caches, and non-blocking caches
- Reducing miss penalty – critical word first, merging write buffers, and prefetching
- Reducing miss rate – compiler optimizations and prefetching

Small and simple L1

- Keeping L1 small with low associativity reduces hit time and energy consumption
- Use CACTI to determine access time and energy per access
  - Cache modeling tool developed by HP Labs
  - Norman P. Jouppi was a major contributor

Advanced Cache Optimizations

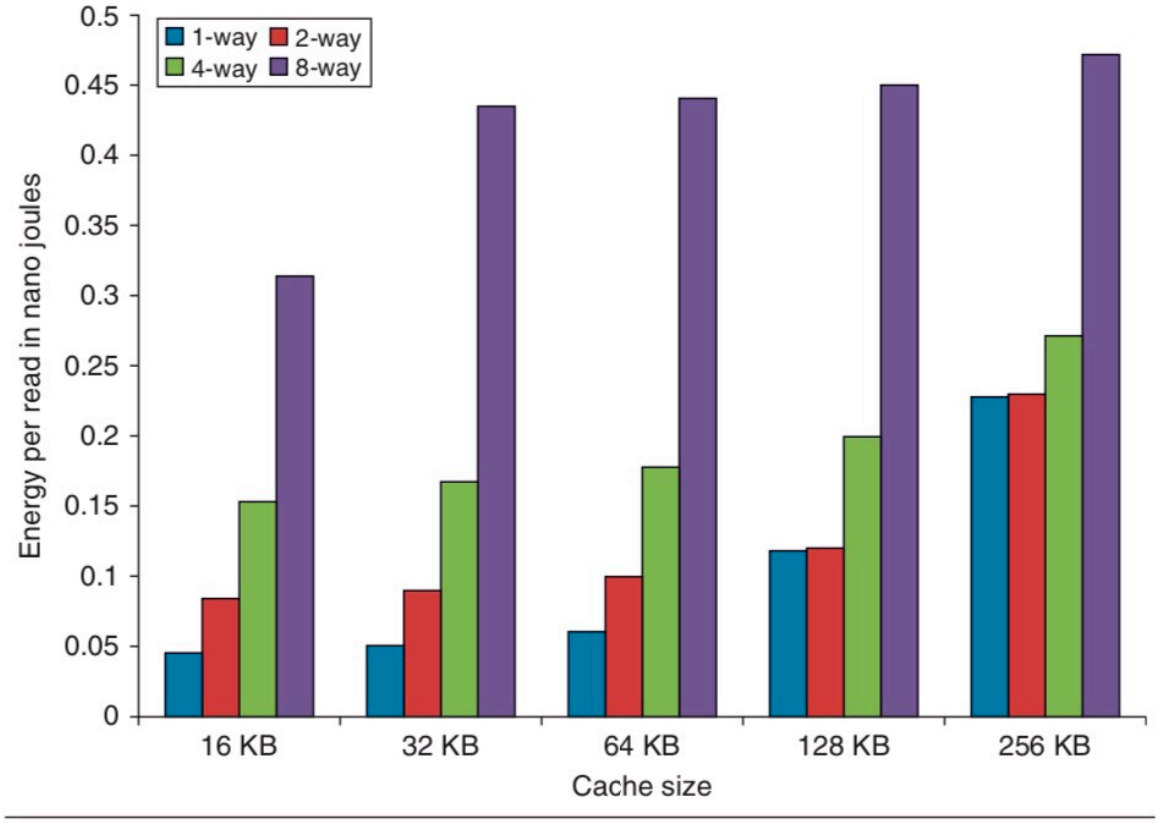


Access times generally increase as cache size and associativity are increased.

Observations:

- From direct-mapped to 2-way set associative, the access time does not change significantly
- There is an anomaly at 64kB such that 2-way set associative has a faster access time than direct mapped. Why might this be?
- There is a significant jump in access time from 4- to 8-way

Advanced Cache Optimizations



Energy consumption per read increases as cache size and associativity are increased

Observations:

- There is a significant jump from 4-way to 8-way
- Cache size has less impact on energy from 16KB to 64KB
- CACTI model assumed single SRAM banks but multi-banking is option to reduce energy allowing only small memories to be accessed at a time

Advanced Cache Optimizations

- Way prediction to reduce hit time
  - Can decrease hit time of associative caches by predicting the way
  - Popular in 2-way set associative caches
  - First used on the MIPS R10000
  - Simulations have shown 90% accuracy for 2-way and 80% for 4-way
  - Must keep extra prediction bits with each cache block to provide a hint as to which way to select on the next access
  - If prediction is incorrect, the processor is stalled and the next cycle is used to recover the correct way
  - Way prediction can be used to save power consumption

    -- Initially only access and compare one tag

    -- Only read one cache block

Advanced Cache Optimizations

- Pipelining caches to increase cache bandwidth
  - At the L1, improve block rate of processor
  - The Pentium had single cycle L1 access, the Pentium Pro to Pentium III had two pipeline stages for L1, the Pentium 4 and beyond have four pipeline stages!
  - Pipelining the instruction cache exacerbates control hazards by adding stages between fetch and branch resolution
  - Pipelining the data cache exacerbates data hazards by increasing the load delay
  - Modern processors solve the control hazard issue with accurate branch prediction and the data hazard issue with out-of-order execution

Advanced Cache Optimizations

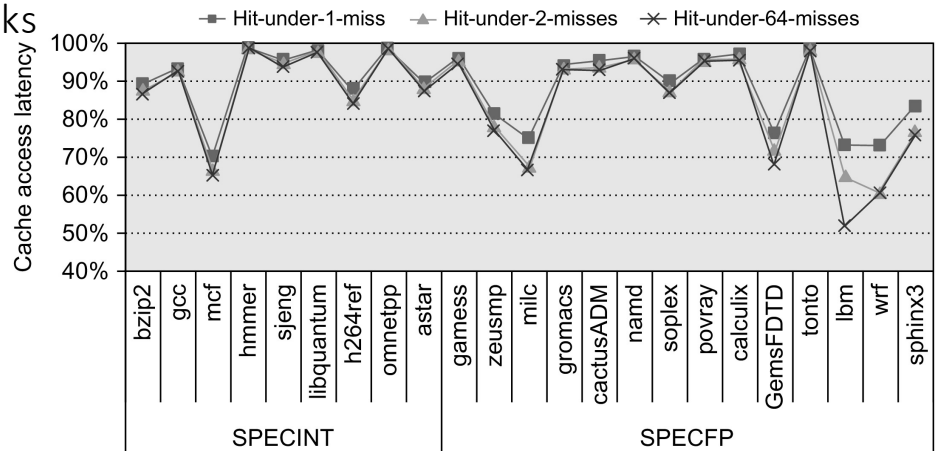
- Nonblocking caches to increase cache bandwidth
  - Out-of-order execution allows instructions following a stalled load to execute
  - Some of these instructions could be another load or a store!
  - A nonblocking data cache will continue to supply hits during a miss
  - Increasing the effectiveness of the out-of-order scheme

    - *How effective is a nonblocking cache?*
    - *Could we allow more than one outstanding miss?*
    - *If so, how many outstanding misses should we allow?*

Advanced Cache Optimizations

Nonblocking Caches

- Farkas and Jouppi investigated these questions in 1994
  - ✓ 8kB cache using SPEC92
  - ✓ 20% reduction in miss penalty for integer benchmarks
  - ✓ 30% for floating point with hit-under-1-miss
- Li et al. reinvestigated in 2011 using modern parameters with SPECCPU2006
  - ✓ Intel i7 with a 32kB L1 with a four-cycle access time
  - ✓ A 256kB L2 with a 10-cycle access time
  - ✓ a 2MB L3 with a 36-cycle access time
  - ✓ All caches are 8-way with 64 byte blocks
    - 9% reduction in access time for integer
    - 12.5% for floating point



Advanced Cache Optimizations

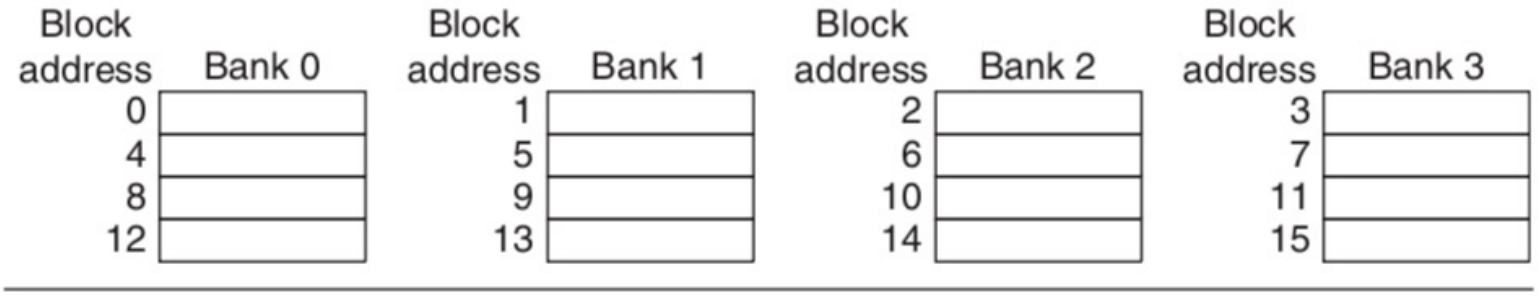
Nonblocking Caches

- Allowing hit-under-2-misses reduces access time by 10% and 16%
- There is little additional benefit to hit-under-64-misses !
- Challenges implement a nonblocking cache
  - *Hits and misses can now collide, e.g. consider a hit to a block that is being replaced in a miss*
  - *Misses can collide with other misses!*
  - *Need a way to keep track of outstanding misses using Miss Status Handling Registers (MSHRs)*

Advanced Cache Optimizations

Multi-banked Caches

- Divide the data into banks using sequential interleaving
- DRAM does this to increase bandwidth in main memory
- For L1 data cache, banking allows multiple cache accesses
- Intel Core i7 has four banks to support up to two memory accesses per cycle



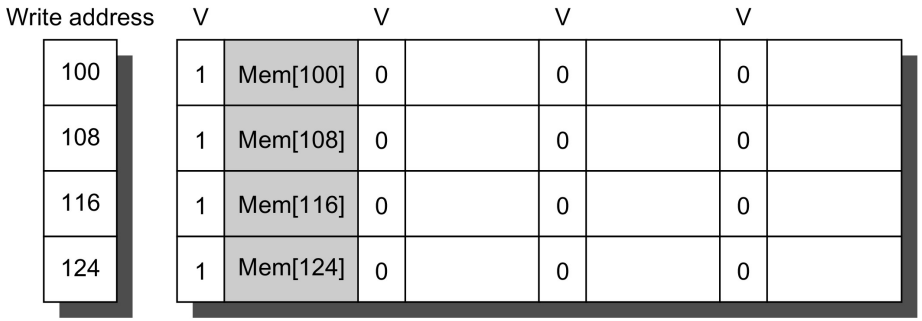
Four-way interleaved cache banks using block addressing

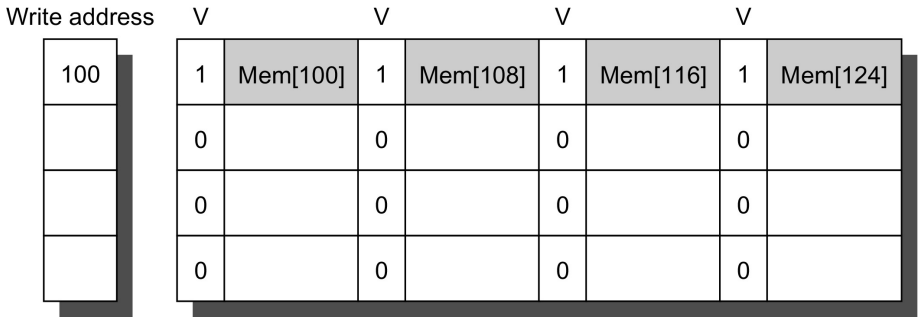
- For L2 and beyond, banking provides support for multiple miss requests and higher bandwidth
- Banking is effective at reducing power! *Why is that again?*

Advanced Cache Optimizations

- Reducing miss penalty
  - *Critical word first* – request the word that caused the miss first and deliver it to the processor ASAP. Continue filling the remainder of the block while the processor continues
  - *Early restart* – request the cache block in normal order but deliver the missed word as soon as it arrives to unblock the processor ASAP

- Merging Write Buffer
  - A **naive approach** to a write buffer is to store each write in an entry without awareness of other entries
  - Inefficient use of bandwidth
  - A **better approach** is to merge writes into previous entries to take advantage of bandwidth





Advanced Cache Optimizations

- Compiler Optimization to Reduce Miss Rate

  - Loop Interchange

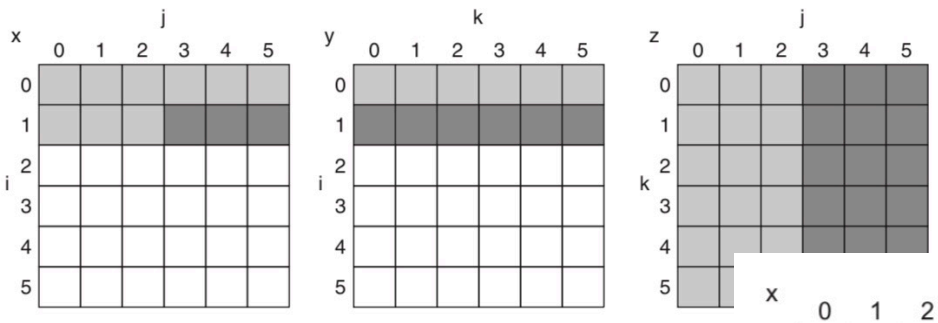
```
/* Before */
for (j = 0; j < 100; j = j+1)
       for (i = 0; i < 5000; i = i+1)
              x[i][j] = 2 * x[i][j];
```

```
/* After */
for (i = 0; i < 5000; i = i+1)
      for (j = 0; j < 100; j = j+1)
             x[i][j] = 2 * x[i][j];
```

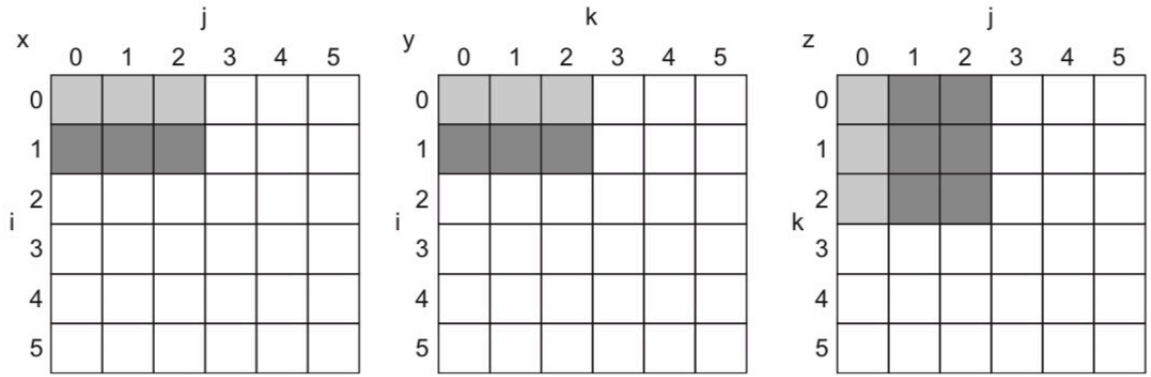
Advanced Cache Optimizations

- Compiler Optimization to Reduce Miss Rate
  - Blocking

```
/* Before */
for (i = 0; i < N; i = i+1)
       for (j = 0; j < N; j = j+1)
              {r = 0;
               for (k = 0; k < N; k = k + 1)
                    r = r + y[i][k]*z[k][j];
               x[i][j] = r;
              };
```



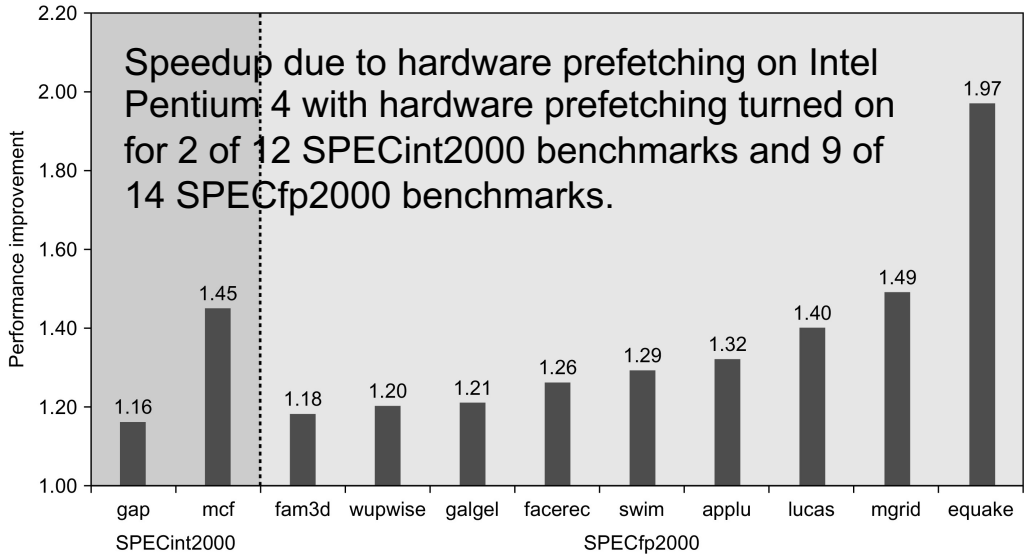

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
       for (j = jj; j < min(jj+B,N); j = j+1)
            {r = 0;
             for (k = kk; k < min(kk+B,N); k = k + 1)
                  r = r + y[i][k]*z[k][j];
             x[i][j] = x[i][j] + r;
            };
```



Advanced Cache Optimizations

- Hardware Prefetching of Instructions and Data
  - Fetch a block before it is used assuming it will be used (spacial locality)
  - Store block in buffer to avoid polluting the cache
  - Hardware vs. Software
    - Hardware prefetching speculatively fetches next blocks
    - Software prefetching uses hints from compiler to determine which blocks to prefetch – **this requires extra prefetch instructions**

- Pentium 4 with hardware prefetching turned on
- Only benchmarks had higher than 15% improvement shown

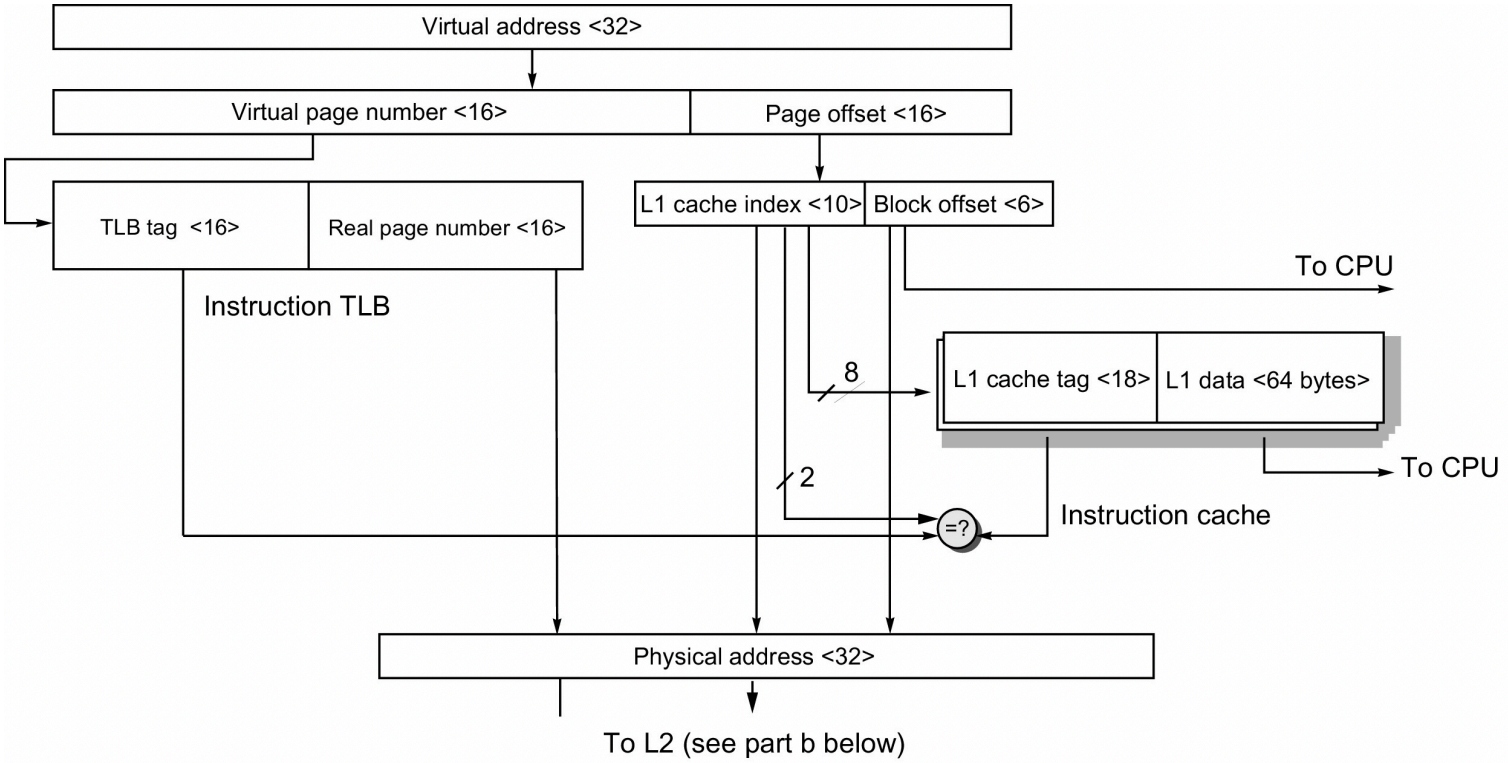


Advanced Cache Optimizations

- Compiler-Controlled prefetching to reduce miss rate penalty or miss rate
  - Register Prefetch: load the value into a register
  - Cache Prefetch: load data only into the cache and not the register

Real World Examples of Memory

- ARM A53 memory hierarchy
  - 64-bit datapth, 8-stage, ***in-order*** pipeline
  - 2-wide Superscalar with floating point unit
  - The little core in Big/Little core architectures



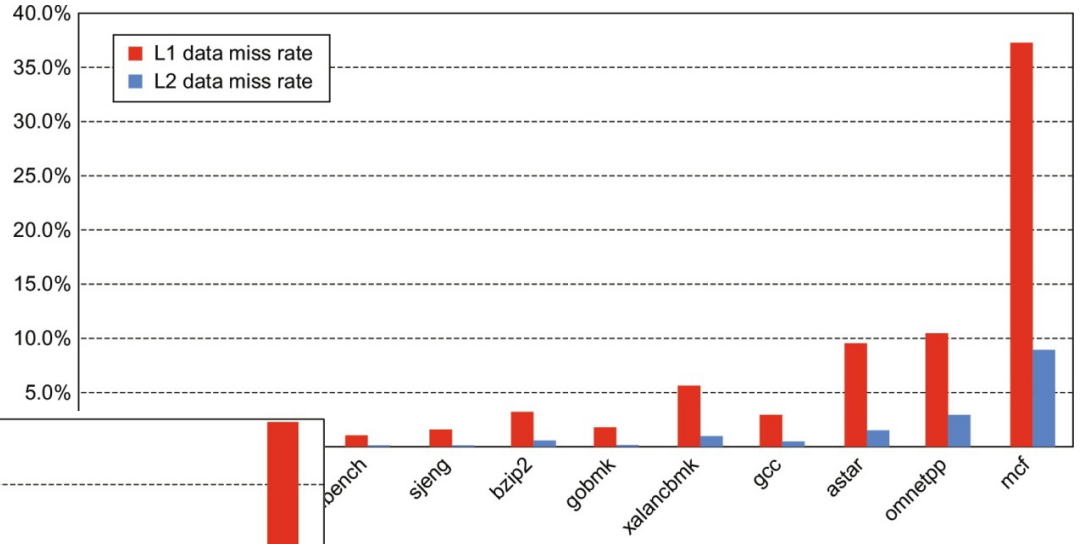
Real World Examples of Memory

- ARM A53 memory hierarchy
  - ✓ A configurable soft IP core
  - ✓ Used in PMDs and other battery powered devices
  - ✓ Clock rates up to 1.3GHz
  - ✓ Uses critical word first
  - ✓ Non-blocking cache allowing hits under 1 miss
  - ✓ L1 caches are virtually indexed, physically tagged
  - ✓ L2 is physically addressed
  - ✓ L1 and L2 use write-back and default to allocate on write

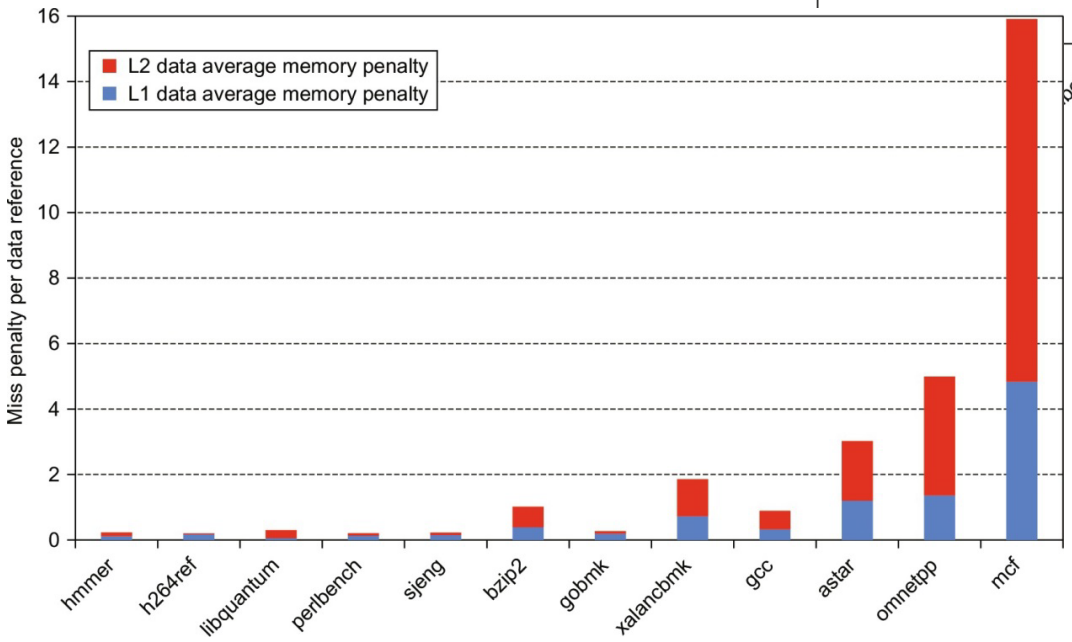
Structure	Size	Organization	Typical miss penalty (clock cycles)
Instruction MicroTLB	10 entries	Fully associative	2
Data MicroTLB	10 entries	Fully associative	2
L2 Unified TLB	512 entries	4-way set associative	20
L1 Instruction cache	8–64 KiB	2-way set associative; 64-byte block	13
L1 Data cache	8–64 KiB	2-way set associative; 64-byte block	13
L2 Unified cache	128 KiB to 2 MiB	16-way set associative; LRU	124

Real World Examples of Memory

- ARM A53 memory hierarchy



- SPECInt2006 L1 local and L2 global
- Instruction miss rates all below 1%
- Local L2 miss rate is 15.1%



- L2 stalls dominate due to higher miss penalty in spite of having a much lower miss rate!

Real World Examples of Memory

- Intel i7 6700 (Click Here)
  - 64-bit datapth, 16-stage, out-of-order pipeline
  - 4-wide Superscalar with multiple floating point and integer units
  - On the aggressive end of the spectrum
    - ✓ Intel's performance desktop line of processors
    - ✓ 6th generation known as the Skylake microarchitecture
    - ✓ In turbo boost mode, it can run at 4.0GHz
    - ✓ Hard IP processor fabricated at Intel

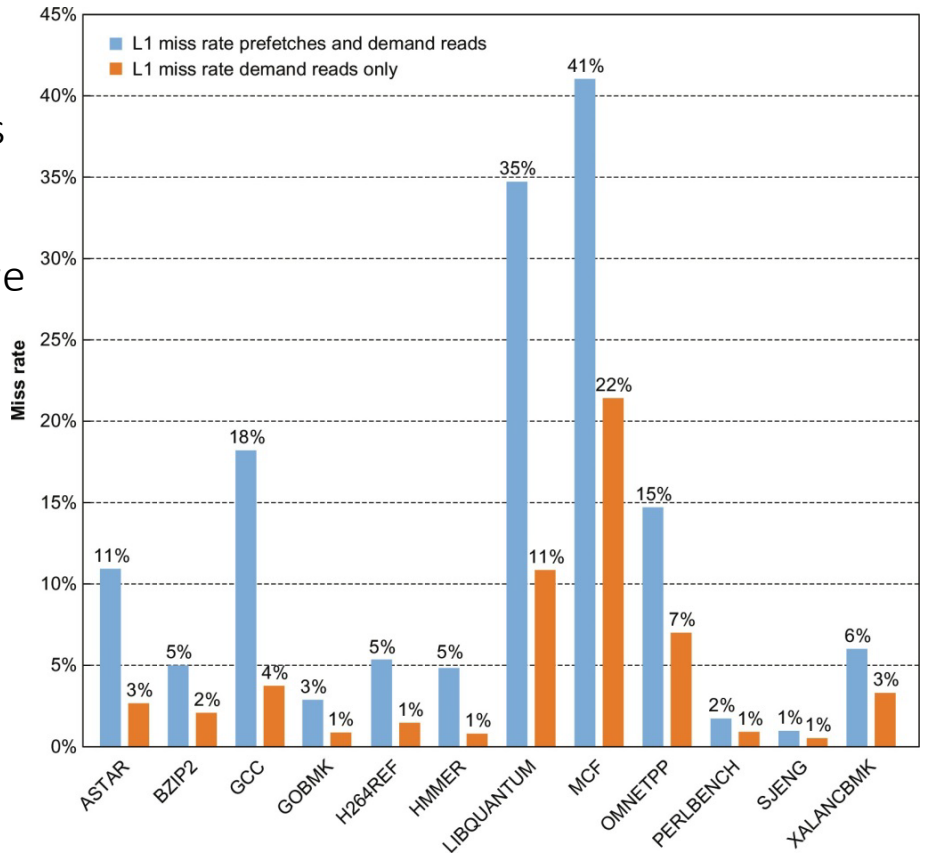
- L1 TLB supports 4KiB and large 2-4MiB pages
- L2 TLB only supports 4KiB pages!
- L2 TLB can handle two misses in parallel
- All caches use write-back and 64-byte blocks
- All caches are non-blocking, allowing multiple misses
- A merging write buffer sits between L1,L2
- L3 is inclusive to L1 and L2

Characteristic	Instruction TLB	Data DLB	Second-level TLB
Entries	128	64	1536
Associativity	8-way	4-way	12-way
Replacement	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU
Access latency	1 cycle	1 cycle	8 cycles
Miss	9 cycles	9 cycles	Hundreds of cycles to access page table

Characteristic	L1	L2	L3
Size	32 KiB I/32 KiB D	256 KiB	2 MiB per core
Associativity	both 8-way	4-way	16-way
Access latency	4 cycles, pipelined	12 cycles	44 cycles
Replacement scheme	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU but with an ordered selection algorithm

Real World Examples of Memory

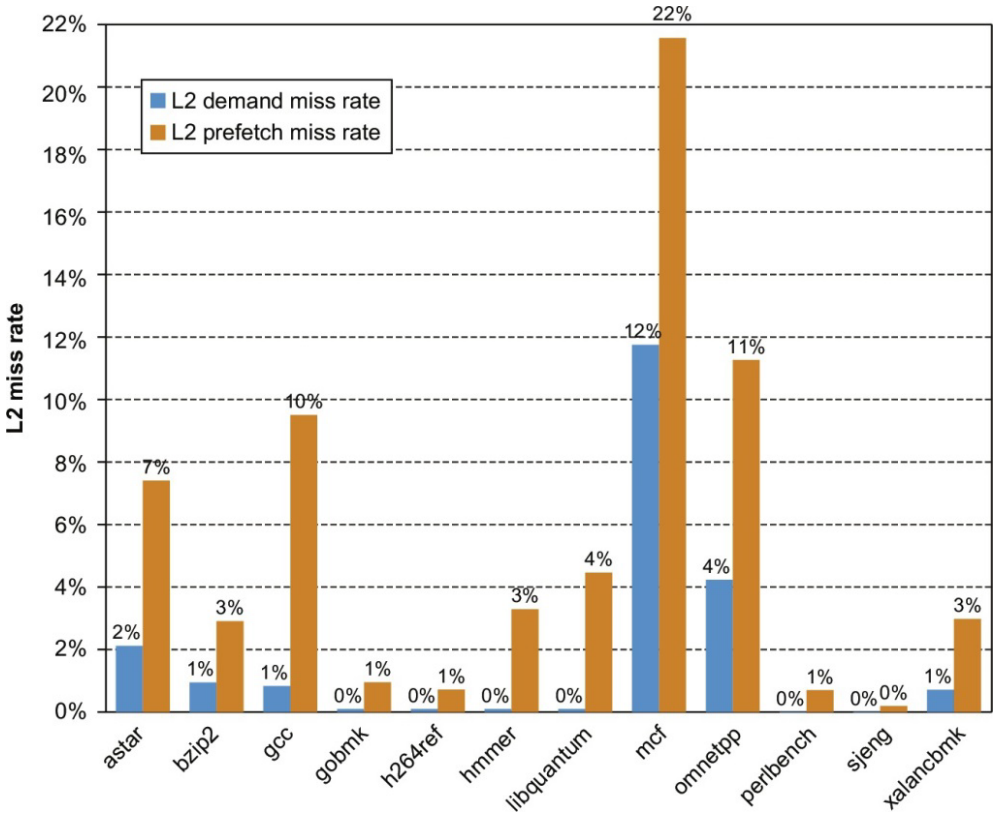
- Intel i7 6700 Memory Hierarchy
  - Notable features (***[Ref]*** Figure 2.25 of the textbook)
  - ✓ 3 channels to DDR3-1066 supports up to 25 GB/s
  - ✓ A distributed L3 with 2MiB per core
  - ✓ L1 caches have 8 banks
  - ✓ L2 caches have 4 banks
  - ✓ L3 has a total of 16 banks across 4 cores



Real World Examples of Memory

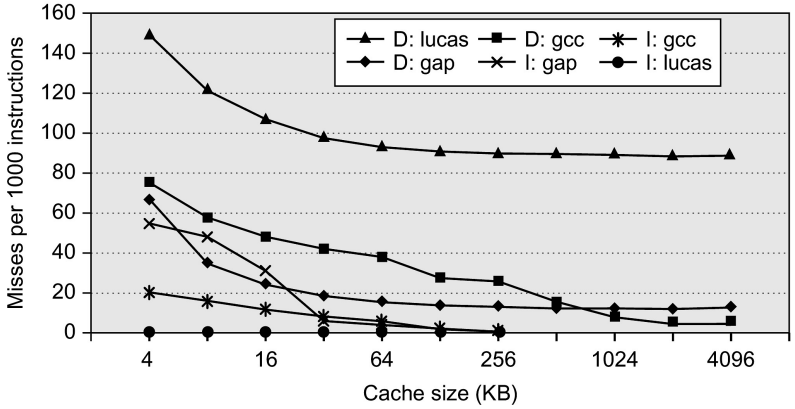
- Intel i7 Miss Rate

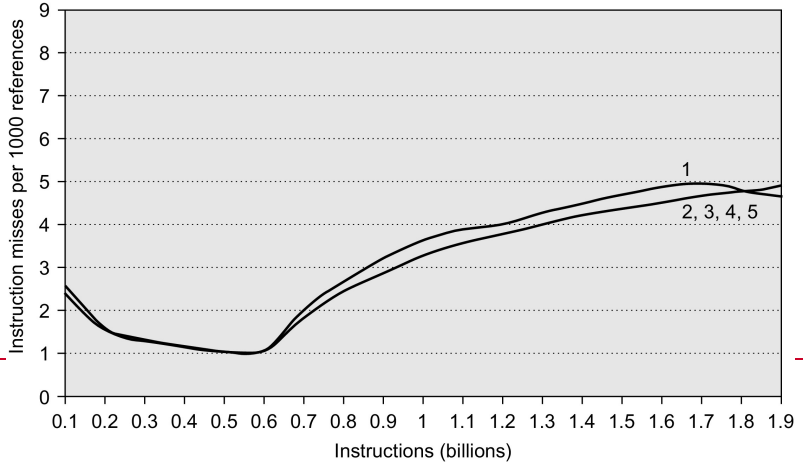
- Miss rate including prefetching
  - ✓ Much higher miss rate but has misses that may not be used
  - ✓ Forces access to L2 prior to need
- Demand only miss rate
  - ✓ Lower miss rate because some data has been prefetched!
  - ✓ mcf is not very cache friendly!

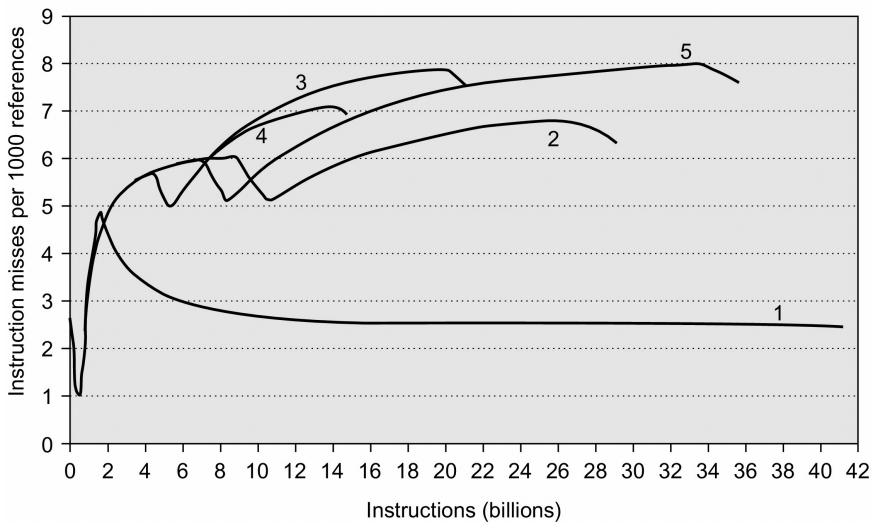


Real World Examples of Memory

- Things to watch out for
  - Cache performance varies significantly from one benchmark to the next!
    - Cache behavior can change wildly throughout program execution
    - **Must** simulate enough instructions to get accurate results
    - **Must** simulated enough benchmarks to get accurate results







- Simulation runs for the perl benchmark from SPEC2000 with five different inputs
- Alpha processor with split L1, 64KiB, 2-way set associative caches and a 1MiB direct-mapped L12

Note of Dynamic Random Access Memory (DRAM)

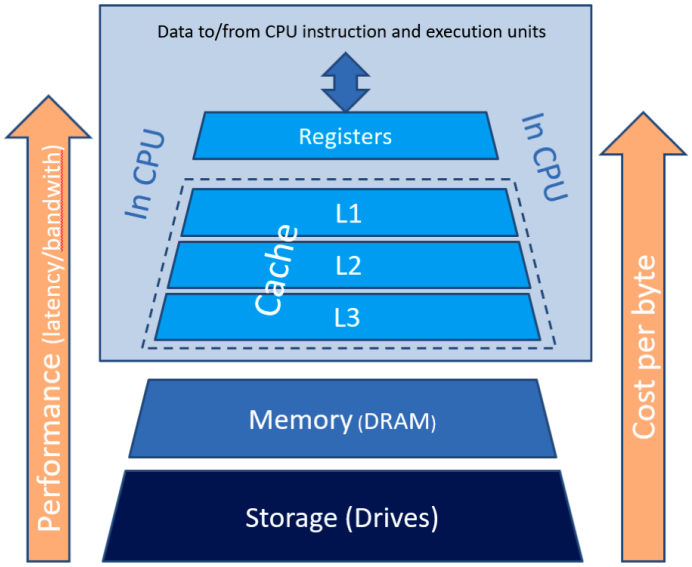
- DRAM is used for main memory, GPU memory, and now L4 caches so it's important that we understand the terms and how it works
  - Dynamic Random Access Memory

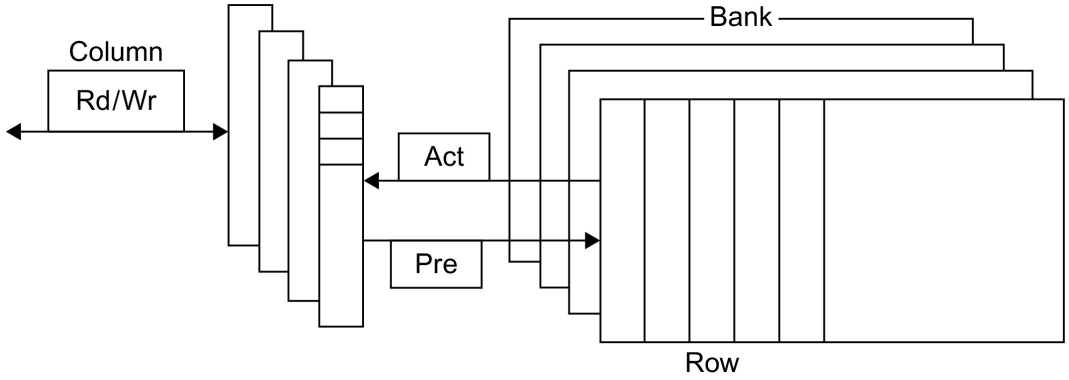
    dynamic implies bits are stored using captured charge
  - Compared to Static RAM which uses cross coupled inverters
  - High density but requires period refresh and must write back row

Click to watch Video

Note of Dynamic Random Access Memory (DRAM)

- DRAM is used for main memory, GPU memory, and now L4 caches so it's important that we understand the terms and how it works
  - DRAM dynamic implies bits are stored using captured charge
  - Compared to Static RAM which uses cross coupled inverters
  - High density but requires period refresh and must write back row

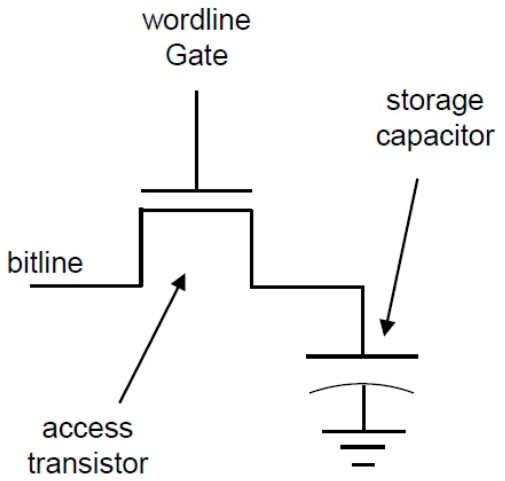


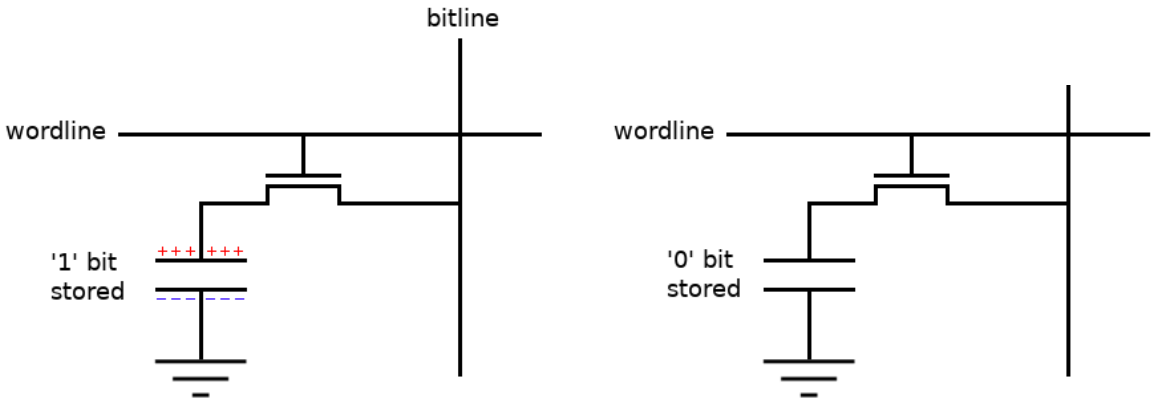


- Modern DRAM is organized in banks, rows, columns
- A bank and row are accessed during the Activate step

Note of Dynamic Random Access Memory (DRAM)

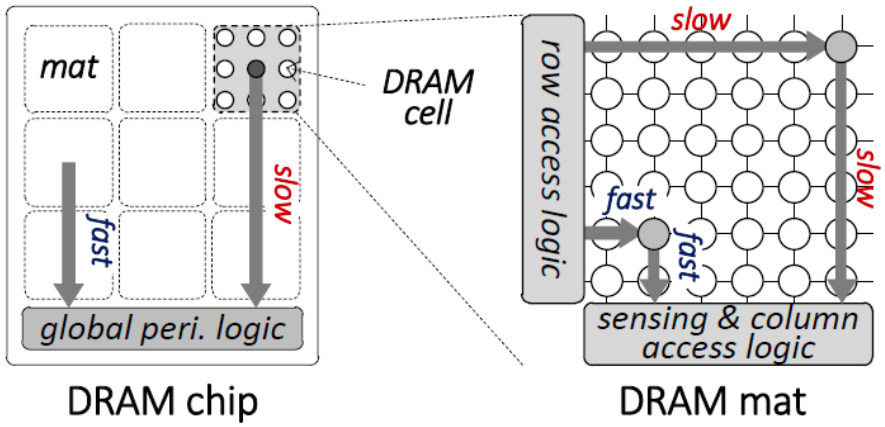
- DRAM Access
  - I/O pins are a precious resource
  - To save I/O, the address is split into a bank, row, and column address
  - I/O pins are the multiplexed such that address is sent to DRAM in parts
  - Data pins are 4-, 8-, or 16-bits in length





NOTE OF DYNAMIC RANDOM ACCESS MEMORY (DRAM)

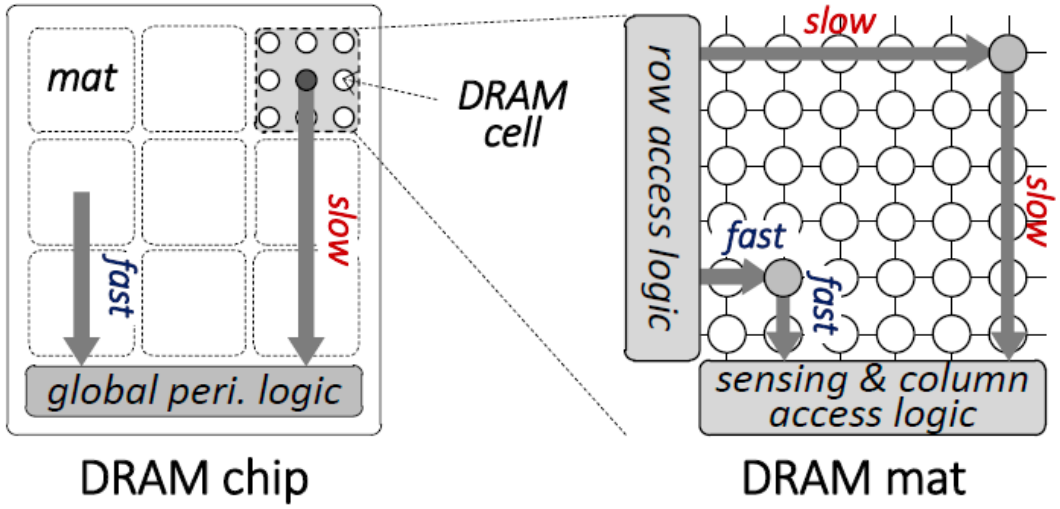
- DRAM Access
  - A row in a particular bank is opened during the Activate step
    - The bit lines are precharged
    - Row decoder enables a row
    - The row data are copied into a row buffer
    - Formerly the Row Access Strobe (RAS) step



  - A column of data within the row is accessed during the CAS step
    - Column Access Strobe (CAS)
    - Uses multiplexing logic to access a portion of the row
    - Column width is 4-,8-, or 16-bit depending on DRAM chip

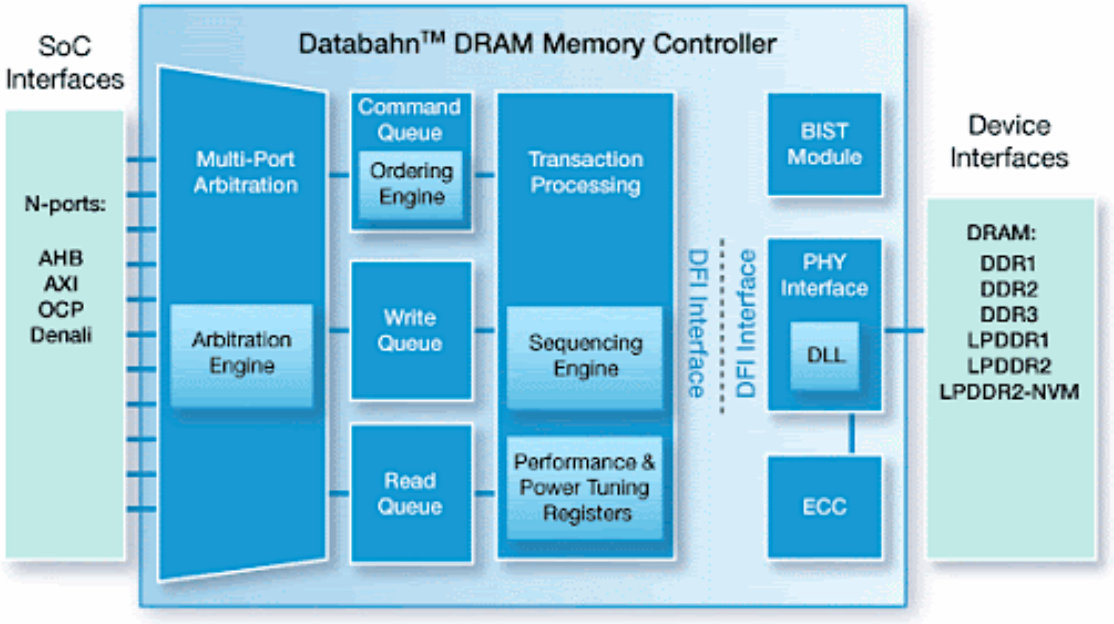
NOTE OF DYNAMIC RANDOM ACCESS MEMORY (DRAM)

- DRAM Access
  - DRAM was formerly asynchronous which required significant overhead built into controller so synchronize with DRAM chip
  - Thus a clock was added and we now call it Synchronized DRAM (SDRAM)
  - To improve bandwidth, we use both edges of the clock, hence the name Double Data Rate (DDR)



NOTE OF DYNAMIC RANDOM ACCESS MEMORY (DRAM)

- DRAM Access
  - After a row is accessed it must be closed which involves writing contents from the buffer back to the row
  - Opening one bank can be overlapped with closing another bank
  - Memory controller attempts to merge accesses to open banks
  - Must also handle periodic refresh around 64ms



NOTE OF DYNAMIC RANDOM ACCESS MEMORY (DRAM)

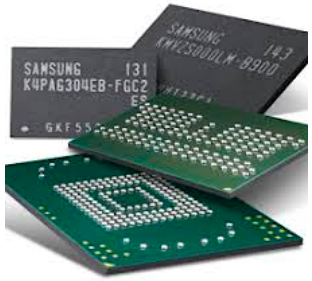
			Best case access time (no precharge)			Precharge needed
Production year	Chip size	DRAM type	RAS time (ns)	CAS time (ns)	Total (ns)	Total (ns)
2000	256M bit	DDR1	21	21	42	63
2002	512M bit	DDR1	15	15	30	45
2004	1G bit	DDR2	15	15	30	45
2006	2G bit	DDR2	10	10	20	30
2010	4G bit	DDR3	13	13	26	39
2016	8G bit	DDR4	13	13	26	39

- DDR1 used 2.5V supply
- DDR2 – 1.8V
- DDR3 – 1.5V
- DDR4 – 1.0V
- DDR4 released in 2016 but expected in 2014

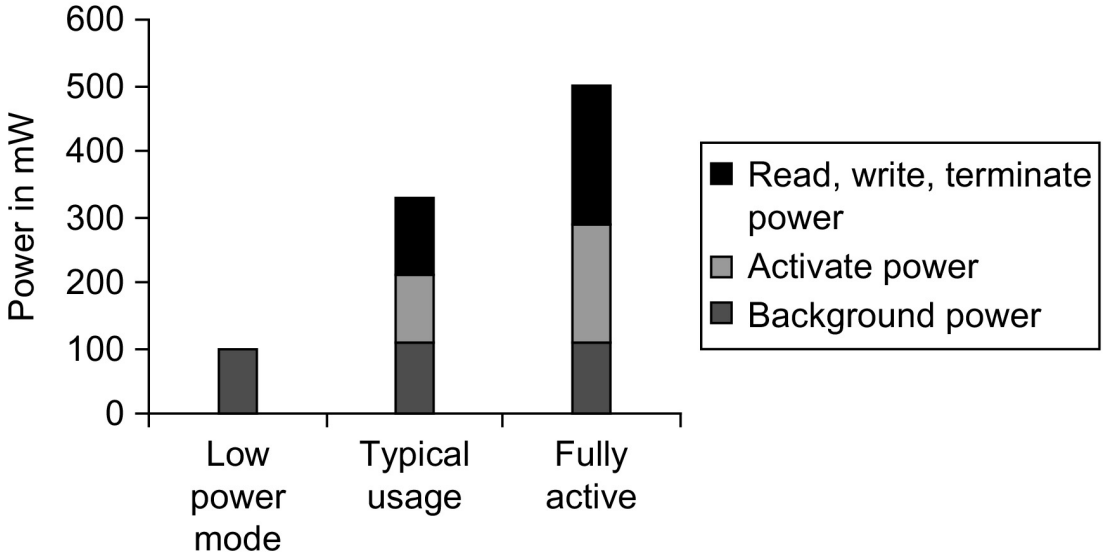
Standard	I/O clock rate	M transfers/s	DRAM name	MiB/s/DIMM	DIMM name
DDR1	133	266	DDR266	2128	PC2100
DDR1	150	300	DDR300	2400	PC2400
DDR1	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1333	2666	DDR4-2666	21,300	PC21300

- DRAM chips are organized in dual inline memory modules (DIMMs)
- 4-16 chips per DIMM providing 8 byte wide interface

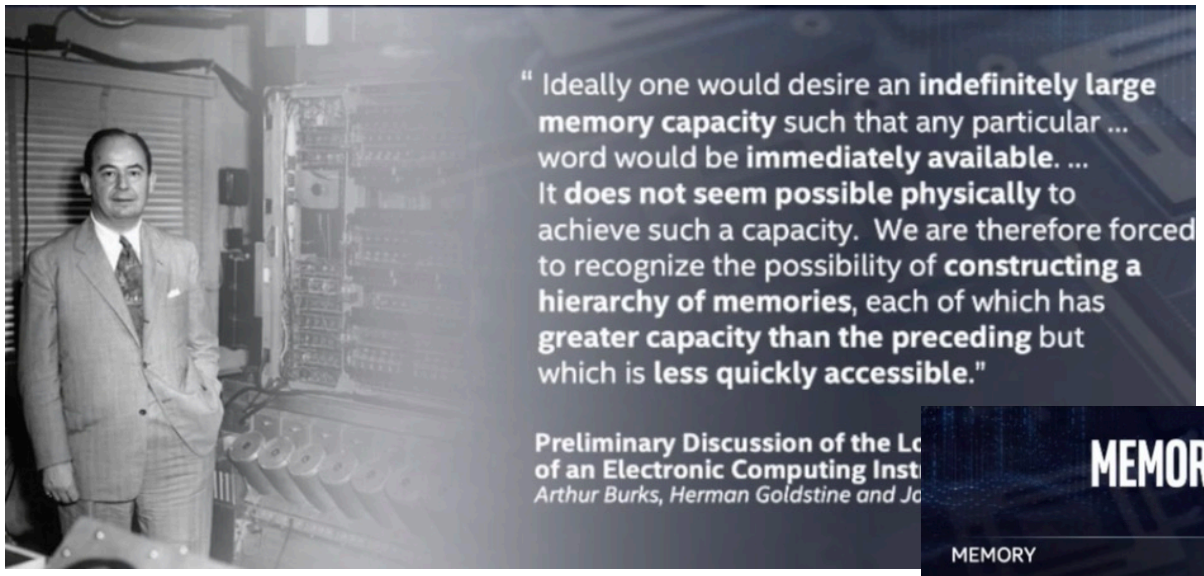
NOTE OF DYNAMIC RANDOM ACCESS MEMORY (DRAM)

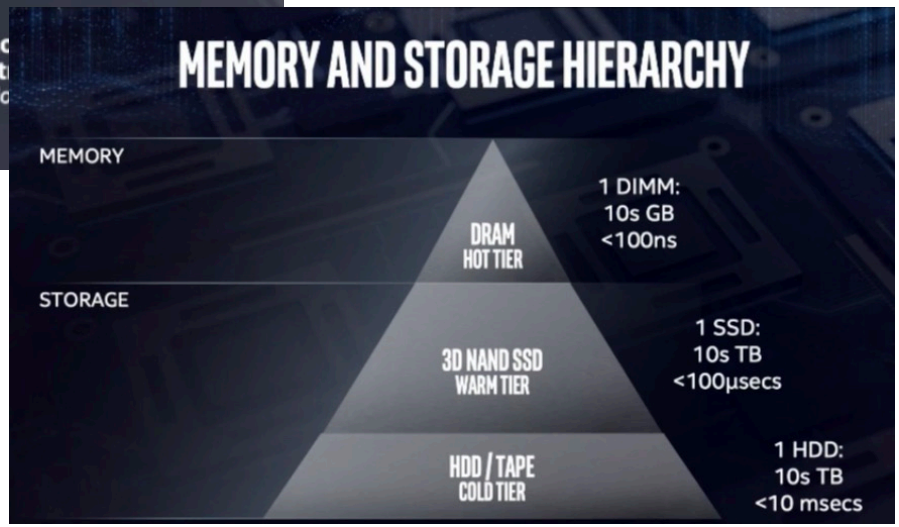


- Power of DDR3 SDRAM
- low-power mode only refreshes DRAM to maintain content



Converging Memory and Storage

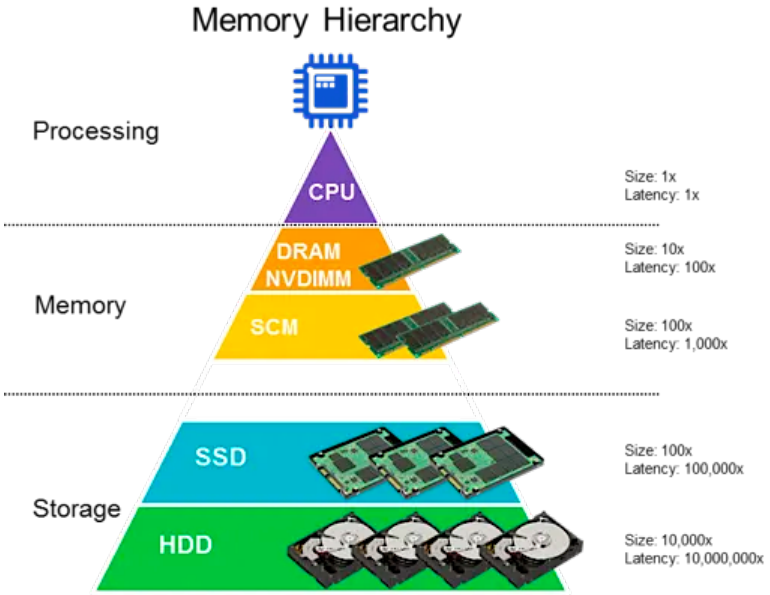




[Click the figure to watch video]

What did we Learn So far

- Introduction of Memory Hierarchy
- Optimization of Cache performance
- Memory Technology and optimizations
- Design of Memory Hierarchy
- Virtual Memory
- Cache optimization
- Real-world Memories



Computer Architecture and Data Storage

David Patterson